UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA AERONÁUTICA Y DEL ESPACIO

GRADO EN INGENIERÍA AEROESPACIAL

Trabajo Fin de Grado

UAV-based Object Tracker with a Convolutional Neural Network Detector and a Kalman Filter-Dynamic Association Tracker

Autor: ABRAHAM LÓPEZ DÍAZ

Especialidad: CIENCIA Y TECNOLOGÍA AEROESPACIAL

Tutor Académico: JOSÉ LUIS PÉREZ BENEDITO

Tutor Profesional: LUIS IZQUIERDO MESA

Junio 2018

Acknowledgements

To the Μοιρες, who caused my ship to ran aground in these shores.

Abstract

Artificial Intelligence and specially Computer Vision is experiencing a dramatic growth, with applications of these technologies in the aerospace field. in this work, an Artificial Intelligence object tracker was trained to track people from a UAV. Chosen approach is tracking-by-detection with YOLO (Convolutional Neural Network object detector) and DeepSORT (Kalman Filter, Dynamic Association tracker). Python implementations of these models were used with the Tensorflow framework. Training dataset UAV123 was improved and YOLO model fine-tuned with this improved dataset. Preliminary systems integration and performance analysis were conducted.

Contents

Chapter 1

Introduction

1.1 Objective

Autonomous target tracking using a drone is the objective of this work. Traditional tracking methods such as GPS beacons or basic image processing are rejected in favour of Deep Learning Computer Vision techniques.

As part of a new line of investigation, all work done is extensively documented. Training process and performance evaluation were conducted in order to assess how feasible this objective is with this technology, and to propose improvements to the system.

Identification of people via Artificial Intelligence is a booming industry with many applications in advertising, criminal prosecution, identity confirmation and security. In China, face recognition algorithms are used to detain criminals (a man was recently found among a crowd of 60,000 people in a festival[1]), and internet retail giant Alibaba now allows to confirm a payment with a face scan[2], using Artificial Intelligence. Defence applications need little introduction, with this technology improving drone automation and reducing the need for human intervention.

1.2 Definitions

Concepts such as "Artificial Intelligence", "Machine Learning" and "Deep Learning" are difficult to define with precision, which should not be surprising, as they are closely related to human intelligence, another yet to fully be described concept. Therefore, as no consensus has been reached simple, practical definitions will be provided here, but should not be taken as a formal definition.

An effort to define these concepts and a thorough history compilation can be found in Stuart Russell and Peter Norvig's *Artificial Intelligence: A Modern Approach*[1], one of the most influential books in the field, used as texbook in hundreds of universities, and the fourth most cited publication of this century, according to the authors. A comprehensive academic perspective on the theoretical foundations of Machine Learning and Artificial Intelligence is the main focus of the book, and should be consulted if the reader wished to dive to that level.

[1] `https://www.bbc.com/news/world-asia-china-43751276`

[2] `https://www.faceplusplus.com/`

Artificial Intelligence

Origins of Artificial Intelligence may be traced to the first efforts to establish a theory explaining human consciousness and intelligence, commonly attributed to the first logician, Aristotle, and his συλλογισμός (syllogisms). Artificial Intelligence remained in the field of logic until the advent of mathematical logic thanks to the work of Bertrand Russel and Alfred North Whitehead in their *Principia Mathematica*. The theoretical foundations by Alan Turing and the developments by von Neumann were the basis of the first modern computers, which finally satisfied the need for vast computation capabilities required by AI.

One of the first major successes of Artificial Intelligence came in 1997 with the victory of Deep Blue over Garry Kasparov, becoming the first computer to beat a world chess champion. Throughout the following years AI began to be used across the technology industry, such as in data mining, industrial robotics, speech recognition, search engines and banking software. This was achieved thanks to decades of innovations like intelligent agents and focusing on specific problems with the highest standards of scientific accountability. However, AI development has not been exempt of criticism.

For example, American cognitive scientist and co-founder of the AI laboratory of the Massachusetts Institute of Technology, Marvin Minsky, argued in his book "It's 2001. Where Is HAL?"[2] that problems such as common sense reasoning were being relegated to a second stage, while most of the research went to commercial applications of neural nets or genetic algorithms. For computer scientist and futurist Ray Kurzweil, the issue is that computer power is not sufficient yet[3], while neuroscientist Jeff Hawkins defends that neural net research ignores the essential properties of the human cortex[4].

On a side note, the previously sci-fi idea of an AI rebelling against its creators is getting more plausible, though such a powerful AI is still decades from here. However, this possibility concerned prominent figures such as Stephen Hawking [3] and Elon Musk, who founded a non-profit research company[4] to investigate safe AI.

Back to the definition problem, one of the earliest definitions was coined in 1956 by John McCarthy[5]: "AI involves machines that can perform tasks that are characteristic of human intelligence". While for the purposes of this work this is enough, it is too broad a definition, where "tasks characteristic of human intelligence" is not defined at all.

Traditionally, algorithms that devote themselves to solve narrow tasks are known as Narrow AI, or Weak AI. All currently Artificial Intelligence systems on Earth are Weak AI. These AI have no self-awareness nor genuine intelligence, but show some characteristics of human intelligence. Siri, Cortana or Alexa are examples of these AI. On the other hand, a Strong AI would be capable of performing any task that a human being can, implying that these machines are capable of experiencing consciousness.

Perhaps a better way of defining Artificial Intelligence is through what AI can achieve. The famous Turing Test is the best example of this paradigm.

> The first version of the game he explained involved no computer intelligence whatsoever. Imagine three rooms, each connected via computer screen and keyboard to the others. In one room sits a man, in the second a woman, and in the third sits a person - call him or her the "judge". The judge's job is to decide which of the two people talking to him through the computer is the man. The man will attempt to help the judge, offering whatever evidence he can (the computer terminals are used so that physical clues cannot be used) to prove his man-hood. The woman's job is to trick the judge, so she will attempt to deceive him, and counteract her opponent's claims, in hopes that the judge will erroneously identify her as the male.
>
> What does any of this have to do with machine intelligence? Turing then proposed a modification of the game, in which instead of a man and a woman as contestants, there was a

[3] https://www.cnbc.com/2017/11/06/stephen-hawking-ai-could-be-worst-event-in-civilization.html

[4] https://openai.com/about/

> human, of either gender, and a computer at the other terminal. Now the judge's job is to decide which of the contestants is human, and which the machine. Turing proposed that if, under these conditions, a judge were less than 50% accurate, that is, if a judge is as likely to pick either human or computer, then the computer must be a passable simulation of a human being and hence, intelligent. The game has been recently modified so that there is only one contestant, and the judge's job is not to choose between two contestants, but simply to decide whether the single contestant is human or machine.
>
> The Turing Test [6]

Machine Learning

Arthur Samuel invented the term "Machine Learning" in 1959[7], defining it as the "ability to learn (a task) without being explicitly programmed". Concerning this work, Machine Learning is simply a necessary way of achieving AI (though not sufficient).

Pattern recognition and computational learning theory gave birth to Machine Learning, which explores the study and construction of algorithms that can learn from and make predictions on data through building a model from sample inputs. Machine learning is employed in a range of computing tasks where designing and programming explicit algorithms with good performance is difficult or infeasible, and has applications such as predictive analytics and computer vision.

Within the field of Machine Learning, there are two main types of tasks[8]: supervised, and unsupervised. The main difference between the two types is that supervised learning is done using a ground truth (what the output values for our samples should be are known a priori). Therefore, the goal of supervised learning is to learn a function that, given a sample of data and desired outputs, best approximates the relationship between input and output observable in the data. Unsupervised learning, on the other hand, does not have labelled outputs, so its goal is to infer the natural structure present within a set of data points.

Common core applications of Machine Learning are classification, regression, density estimation and clustering, the first two being the most important.

- Classification: Identifying to which of a set of categories a new observation belongs, on the basis of a training set of data containing observations whose category membership is known. Classification is an example of pattern recognition and is a supervised learning task, whose corresponding unsupervised procedure is clustering (grouping data into categories based on some measure of inherent similarity or distance).
- Regression, also a supervised problem, is the problem of estimating or predicting a continuous quantity product of the relationship of variables.

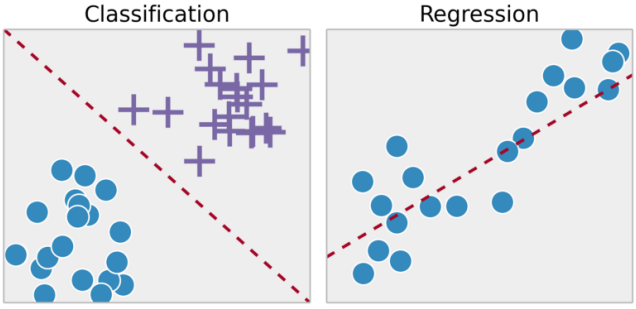


Figure 1.1: Illustration of classification and regression applications. Taken from [8].

Deep Learning

"Deep learning" is just one of many approaches to machine learning, characterized by the use of Artificial Neural Networks with hidden layers. Other approaches include, among others:

- Decision tree learning
- Inductive logic programming
- Clustering
- Bayesian networks
- Genetic algorithms
- Reinforcement learning

1.3 Artificial Neural Networks

Artificial Neural Networks (ANN) are algorithms that mimic the biological structure of the brain. ANN consist of layers that are made of interconnected neurons.

1.3.1 The Neuron

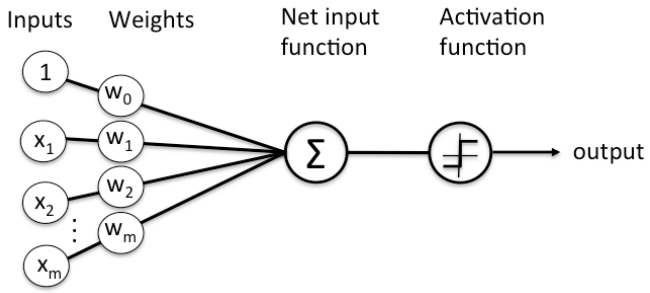


Figure 1.2: Scheme of an artificial neuron. Taken from [9].

A neuron is basically a mathematical operation. It has m inputs that have different weights w_i. Input x_0 is usually assigned the value 1, thus representing a *bias* of value w_0. The sum of the weighted inputs is passed through an activation function φ (also known as transfer function). Output of the neuron is therefore:

$$y = \varphi\left(\sum_{i=1}^{m} w_i x_i + w_0\right)$$

Activation functions

The interesting part of the neuron is the activation function. Non-linear functions are what allows neural networks to produce complex results with a relatively low number of neurons. Linear activation functions are not useful, because multiple layers with these functions are equivalent to a single-layer model. Common activation functions include:

Name	Equation	Order of continuity	Monotonic	Approximates identity near the origin
Identity	$\varphi = x$	C^∞	Yes	Yes
Step function	$\varphi = \begin{cases} 0 & \text{for } x \leq 0 \\ 1 & \text{for } x \geq 0 \end{cases}$	C^{-1}	Yes	No
Sigmoid	$\varphi = \dfrac{1}{1+e^{-x}}$	C^∞	Yes	No
TanH	$\varphi = tanh(x)$	C^∞	Yes	Yes
ReLU*	$\varphi = \begin{cases} 0 & \text{for } x \leq 0 \\ x & \text{for } x \geq 0 \end{cases}$	C^0	Yes	No
Gaussian	$\varphi = e^{-x^2}$	C^∞	No	No

Figure 1.3: Some of the most used activation functions. *ReLU stands for rectified linear unit. Leaky ReLU can be used if the "dying ReLU" problem arises: If the output of ReLU is consistently 0 the gradient is 0, and the error signal backpropagated is nullified. Leaky ReLUs have a small negative slope for $x \leq 0$, solving the problem.

Several activation function attributes such as range, monotonicity and derivative confer different properties to the network, but there are two attributes of special interest:

- Continuously differentiable: This property is desirable (ReLU is not continuously differentiable and has some issues with gradient-based optimization) for enabling gradient-based optimization methods used during training.
- Approximates identity near the origin: When activation functions have this property, the neural network will learn efficiently when its weights are initialized with small random values. When the activation function does not approximate identity near the origin, special care must be used when initializing the weights[10].

Some functions output values between 0 and 1, while other output values between -1 and 1, or even between $-\infty$ and ∞, with the criterion being the task intended (functions that output values between 0 and 1 are preferable for classification). Differences in shape or curvature are secondary to compatibility with optimization procedures.

1.3.2 The Network

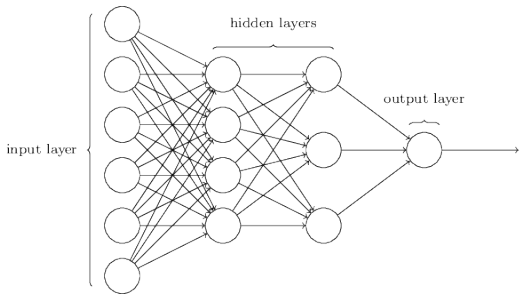


Figure 1.4: Example of a Neural Network with 6 input nodes, 2 hidden layers and an output layer with a single node. Taken from [11].

A network is just a set of connected neurons (also called in this context nodes) arranged in layers, such as in Figure 1.4. The first layer is called "input layer", where the input enters the model. If the model is supposed to process an image, a neuron for each pixel of the image is required. The last layer is called "output layer" and, as the input layer, output dimension constrains the size of this layer. If the model is a single class classifier, output layer will consist on a single neuron, which will usually output a value between 0 and 1 denoting the confidence that the input image contains an instance of the class. The remaining layers are called "hidden layers".

In order to visualize more intuitively what happens inside a NN, an example is provided in Figure 1.5. A possible application of this network would be searching for a number. Each input pixel of the image is weighted differently by each node in the hidden layer. Each node would have weights that optimize the search for different strokes. Output of this layer is passed to the the output layer, which, in this classification task, would output the probability that the input image contains the desired number (usually, if the probability is higher than 0.5 it would be considered as "True" and else "False").

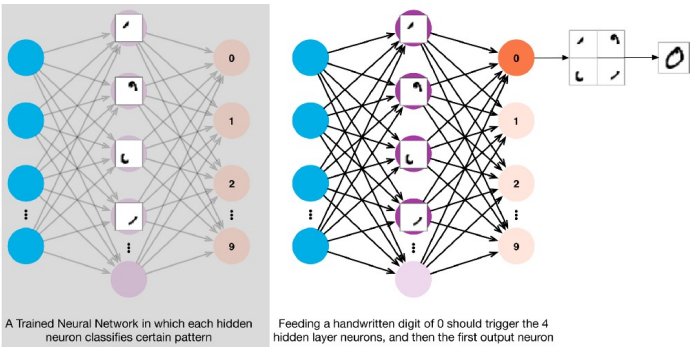


Figure 1.5: Representation of what is happening inside a hidden layer. Taken from [11].

With the addition of more hidden layers, the network progressively captures more complexities of the input, thanks to the work of previous layers. An example of this behaviour is Figure 1.6.

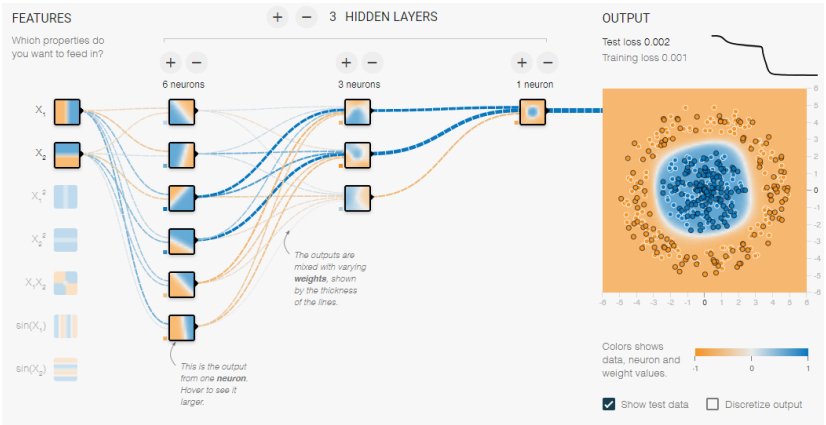


Figure 1.6: Each hidden layer progressively learns more complex patterns. Obtained in Tensorflow Playground, an interactive website where a basic neural network can be intuitively designed by the user and trained to classify simple datasets.

Adding more layers will, on one hand, increase the "intellectual ability" of the network, allowing it to detect more complex patterns more reliably, at the expense of increased computational cost. However, it also increases the ability of the network to memorize. If a big network was built with the intention of recognizing numbers, it would probably just outright memorize them, with no ability to handle small variations in the input. Instead, smaller networks are forced to generalize and "understand" the input in order to produce sensible output.

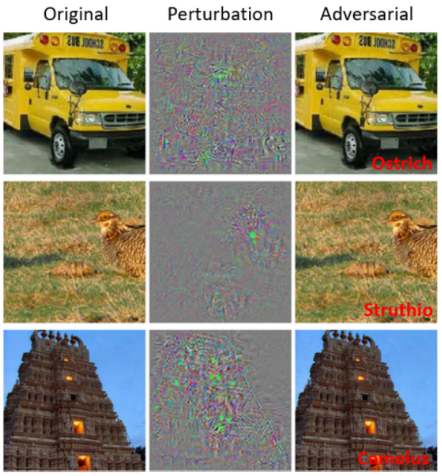


Figure 1.7: Perturbations not seen by the human eye can easily foolish Neural Networks, showing that these models do not have an understanding of vision comparable to humans. Perturbations in this image have been magnified 10x. Taken from [12]

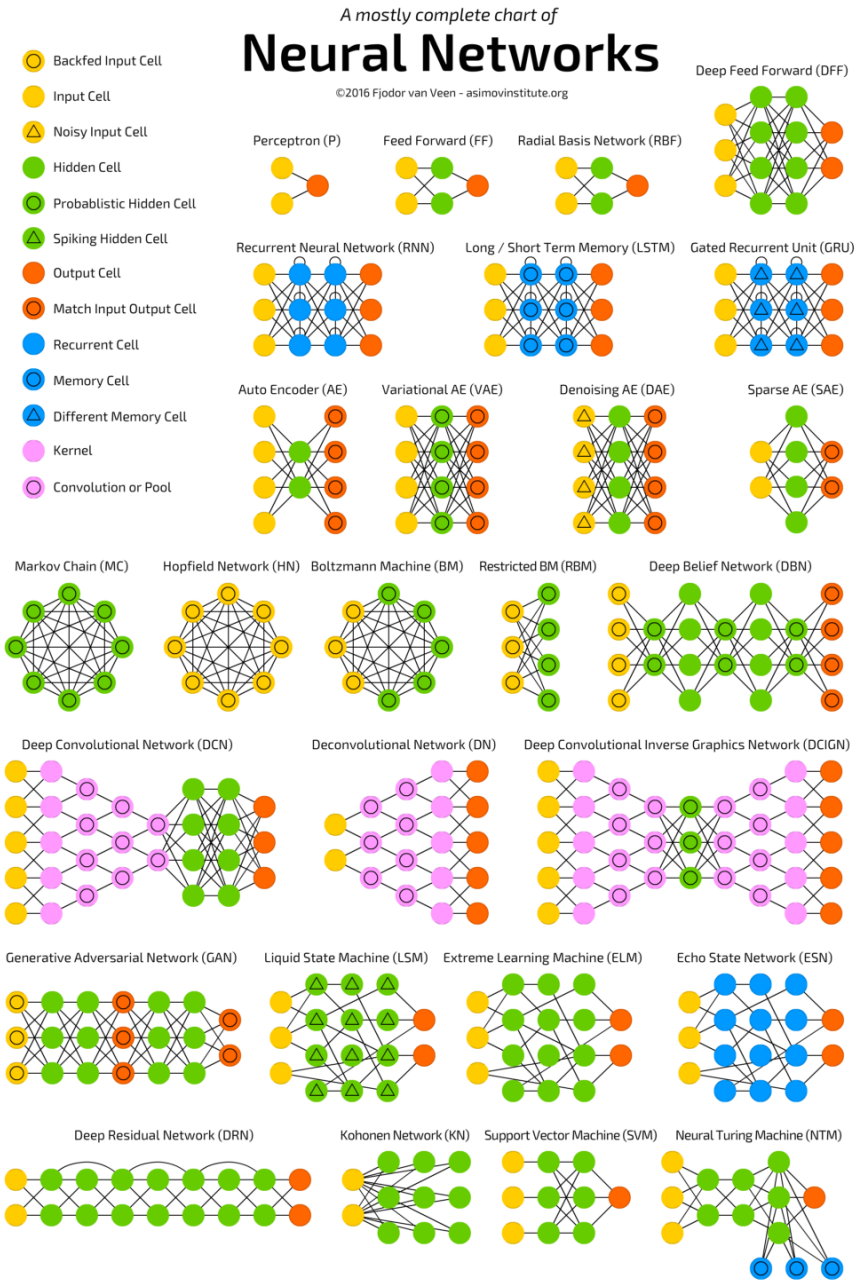


Figure 1.8: Chart showing the structure and special nodes of the most common neural networks. Taken from [13]

However, any insight into what is happening inside a neural network is difficult to visualize and quickly lost even with relatively low network sizes. This is a frequent problem with no solution yet (although there is progress being made[14][15]). Neural networks should be treated as black boxes, and even when the neural network is behaving as desired, it does not mean that it fundamentally understands its task as humans do. An Artificial Intelligence, just as humans normally do, take the shortest paths to success. True understanding is much more difficult to reach than an approximation that works well in the small subset of reality that the AI will be exposed to. It is said that an AI will never perform a task complex than hacking its reward function (which is itself an intelligent strategy, as inconvenient for humans as it can be).

This behaviour induces the AI to be vulnerable in significantly different ways that human intelligence is flawed. An example of this are adversarial attacks (Figure 1.7). These attacks consist of, in the case of computer vision, introducing slight perturbations invisible to the human eye that completely defeat the network. There is currently an accelerating arms race[16] between increasingly effective adversarial attacks and defensive techniques. Recently, NNs were defeated with just a single pixel[17]. Note that humans are not invulnerable to these attacks either[18].

There are many ways to arrange layers and the connectivity between nodes, and modifications to the standard neuron are common and the basis of many remarkable networks. A chart showing most of them is given in Figure 1.8. Some of them will be discussed in the following sections.

1.3.3 Recurrent Neural Networks

In Recurrent Neural Networks (RNN), the output of a node at time t is also an input for that node at subsequent computations (Figure 1.9). This gives these networks a short term "memory". RNN excel at language processing, being the core of almost all speech recognition AI, but also capable of robot control, music and rhythm learning and time series prediction.

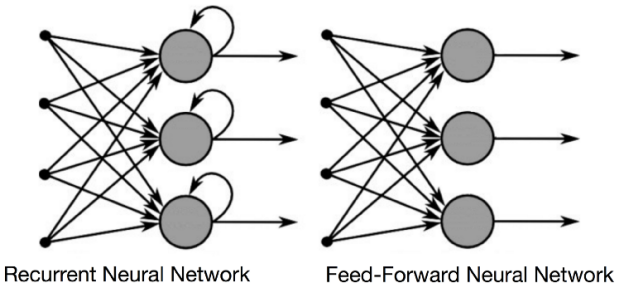


Figure 1.9: Difference between a RNN and a simple Feed-Forward Neural Network. Taken from [19].

The memory of these networks is short lived, spanning no more than 10 previous states. However, this is solved with the introduction of memory cells with a "forget gate". This gate controls when the stored memory is deleted or substituted, allowing for longer retention.

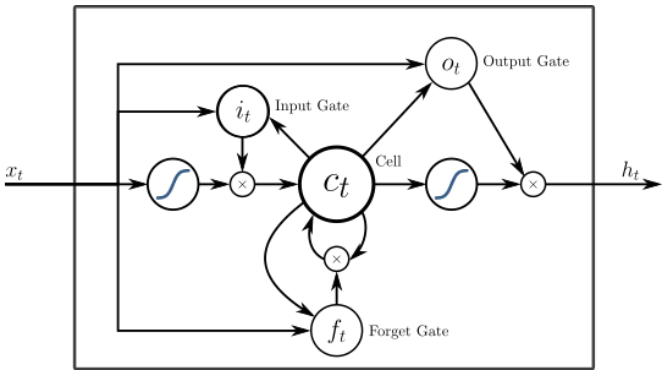


Figure 1.10: A Long short-term memory neuron. Taken from [20].

1.3.4 Convolutional Neural Networks

A convolution is an integral that expresses the amount of overlap of one function g as it is shifted over another function f. In Figure 1.12, the blue region indicates the product of $g(\tau) \cdot f(t-\tau)$, while the red curve is the product of the overlap of the two functions at each point along the x-axis.

$$f \otimes g = \int_{-\infty}^{\infty} f(\tau) g(t-\tau) d\tau$$

Figure 1.11: Convolution.

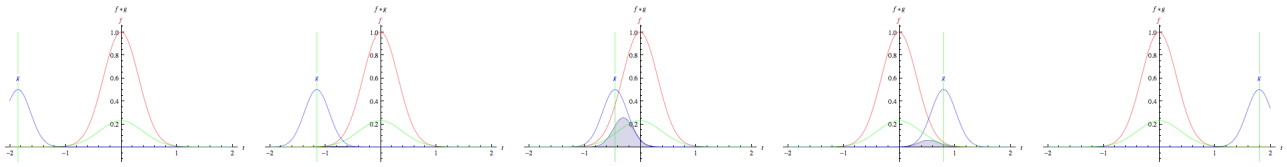


Figure 1.12: Illustration of convolution. The blue bell slides from left to right. . Taken from [21].

Instead of just convolving two simple functions, in Convolutional Neural Networks (CNN) the input (an image, for example) is convolved with several filters. An image can be thought of as a $width \times height \times 3$ matrix (depth 3 because of RGB channels). Filters are normally only a fraction of the width and height (in pixels) of the image, but are 3 layers deep to retain RGB. The output of the convolution of the input and the filter is placed on a "feature map" or "activation map" (which can be reduced to depth 1 by adding together the activation maps of each RGB channel). Examples are given in Figures 1.13 and 1.14.

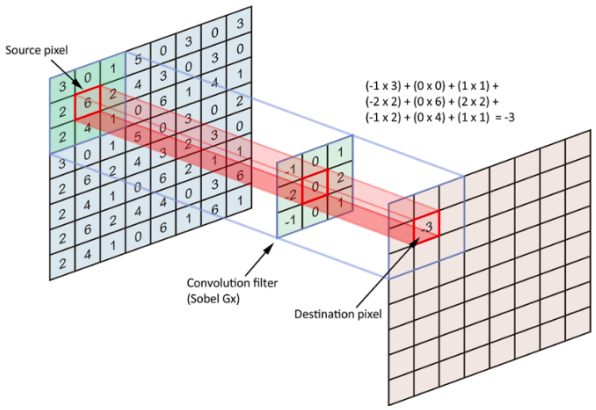


Figure 1.13: Graphic example of a 3x3x1 convolutional filter. Taken from [22].

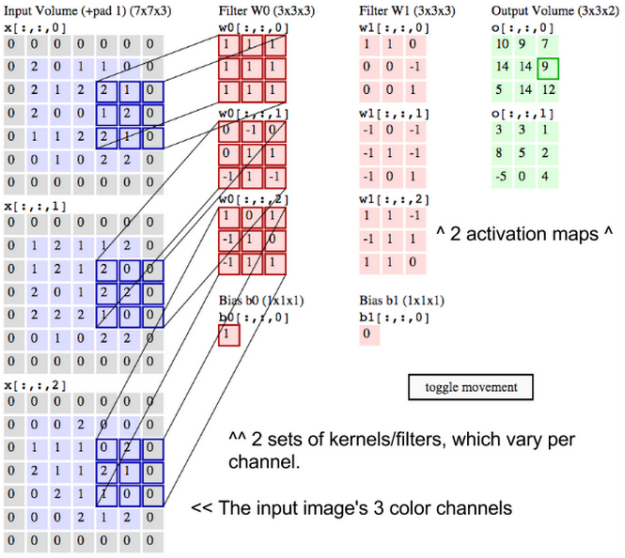


Figure 1.14: Example of a 5x5x3 image convolved with 2 3x3x3 filters. Taken from [25].

After a layer of filters, it is common to place a layer called "pooling", "max pooling", "downsampling" or "subsampling". Usually, this operation consists simply in selecting the largest value for each small section of the image, and the rest are discarded. Only the locations on the image that showed the strongest correlation to each feature (the maximum value) are preserved, and those maximum values combine to form a lower-dimensional space. Information about lesser values is lost in this step, but downsampling has the advantage, precisely because information is lost, of decreasing the amount of storage and processing required.

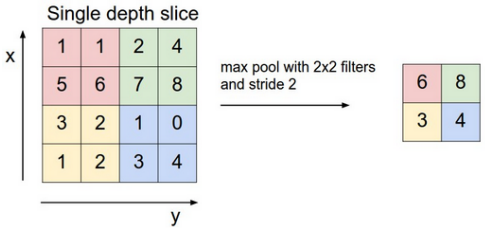


Figure 1.15: Pooling. Taken from [25].

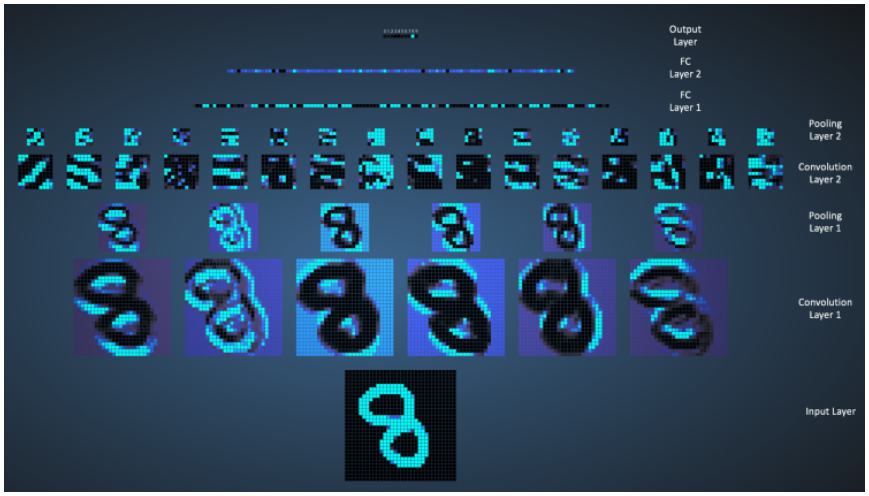


Figure 1.16: Illustration of what happens at each layer of a CNN that is classifying a number in an image. Taken from [23].

CNNs are built alternating filters and pooling layers, placing at the end of the network feed forward layers (the basic type) to, for example, classify the output of the convolutional filters.

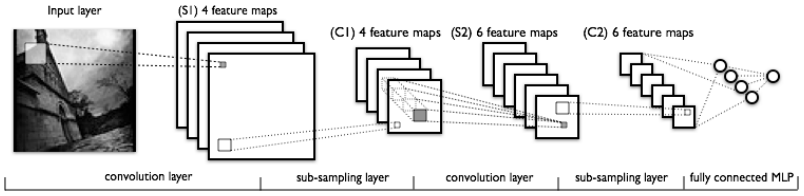


Figure 1.17: Typical structure of a CNN. Taken from [25].

These networks are commonly visualized by arranging convolutional filters in volumes representing their tensor dimensions.

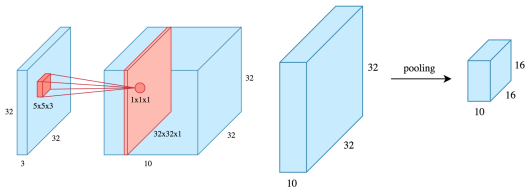


Figure 1.18: Volumetric representation of input and convolutional filters (left) and pooling operation (right). Taken from [24].

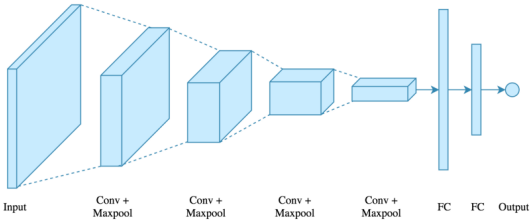


Figure 1.19: Full volumetric representation of a CNN. Taken from [24].

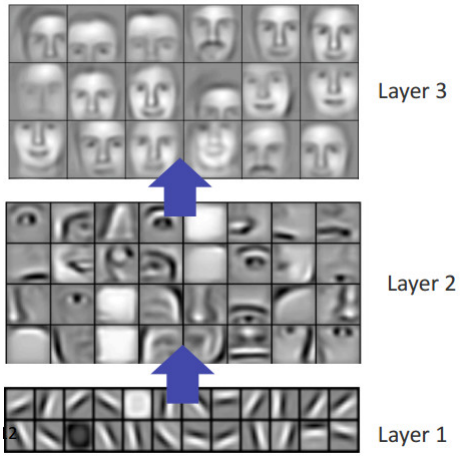


Figure 1.20: Conceptual illustration of how a CNN understands human faces. Taken from [26].

Even though it is not possible to exactly know what is happening at every layer and node of a big neural network that processes complex input, it can be abstracted. For example, Figure 1.20 shows how a CNN may understand images. The first filters, working with the raw image, look for simple strokes

and features, like lines, circles and regions with high contrast. After several layers, the filters may be processing data representing more complex formations, such as an eye or a nose. The last layers work with an structure made of the simpler elements of prior layers.

1.3.5 Generative Adversarial Neural Networks

Generative Adversarial Networks (GAN) are a type of Neural Networks invented by Ian Goodfellow et al. in 2014[27]. In a GAN there are two different neural networks (usually CNNs) where one is a generator and the other a discriminator. The generator is an inverse network that produces data from a random seed while the discriminator is a standard CNN classifier that tries to distinguish if the input belongs to the training set or to the generator, that is, if its true or a forgery. In this zero-sum game the optimization leads to the generator producing increasingly realistic output in order to deceive the discriminator.

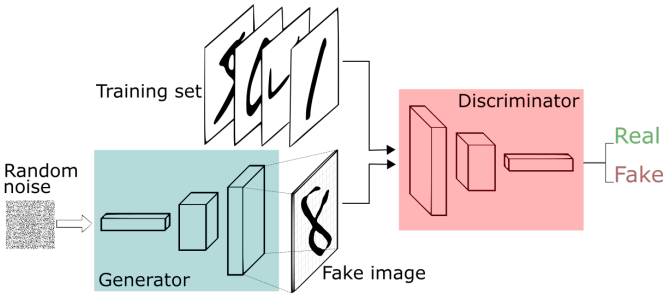


Figure 1.21: Generative Adversarial Network structure. Taken from [28].

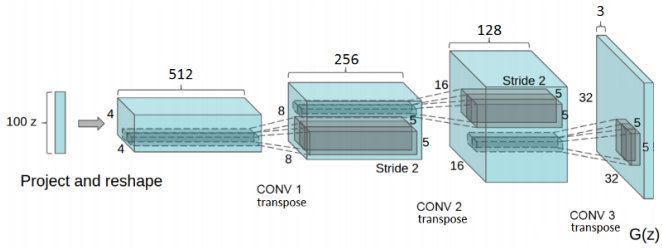


Figure 1.22: Example GAN generator. Note that there are no fully connected nor pooling layers. This type of network is called "Fully Convolutional Networks". Taken from [29].

One of the advantages of this NN is that belongs to the unsupervised learning realm, meaning that no annotation or labelling is required, making for an easier training. These networks have applications mostly in Computer Vision, where they are capable of creating almost incredible new objects and faces[30], interpolate images and improving the quality of low resolution images[31].

Figure 1.23: These faces are not real. They have been created by a GAN. Taken from [30].

The ability to create almost indistinguishably real images[32] and even videos lip synced to a different audio than the original[33] has raised concerns. It is a threat to several aspects of our lives, such as acceptable evidence in trials. As part of ongoing efforts to reliably detect potentially dangerous falsifications, a dataset of forged videos has been created[34].

Figure 1.24: Fake images of Daniel Craig created by Face2Face[32] (left) and fake lip syncing of Obama[33] (right) compared to results obtained by Face2Face.

Figure 1.25: A dataset has been created with forged images to better train defences against this practice. Taken from [34].

1.3.6 Residual Networks

Residual Networks were invented by a team of Microsoft Research[35] in 2015. These networks are built with blocks that output a combination of the raw input and the output of two weight layers, as shown in Figure 1.26.

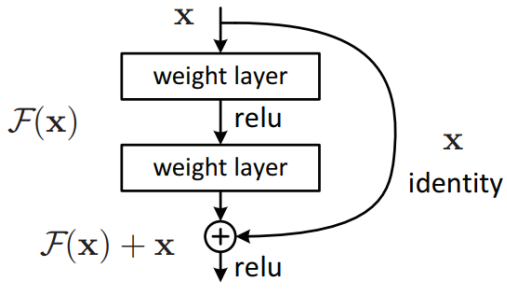


Figure 1.26: The building block of Residual Networks. Taken from [35].

These networks address the problem of very deep networks that experience problems in training (instabilities in the optimization algorithms) and accuracy saturation. Although mitigated by the implementation of regularization and normalization, these issues had no satisfactory solution until this breakthrough. Residual Networks are easier to optimize and gain accuracy with depth, resulting in thin, very deep networks (though subsequent research found wide residual networks to be promising[36]) up to even a thousand layers.

1.4 Computer Vision Tasks

Computer Vision at its core consists in Image Classification: assigning an input image one label from a fixed set of categories[37]. Several other tasks spring from this basic one:

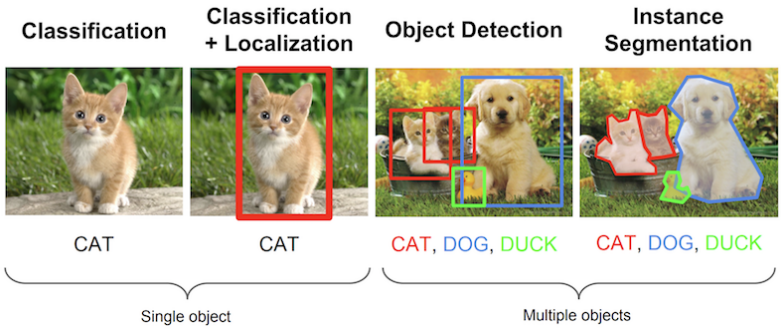


Figure 1.27: Computer Vision basic tasks. Taken from [130].

- Object Localization: It requires obtaining the position of the instance of the object labelled. The box that delineates the object is called "bounding box".
- Object Detection: Same as Object Localization, but with multiple objects.

Before introducing Segmentation, the concepts of "things" and "stuff" must be explained. Things are countable objects, such as people or animals, while stuff denotes amorphous regions of similar texture, such as grass, road or sky. Different algorithms have been traditionally applied to detecting things and stuff, even though they may seem similar tasks at first glance.

- Instance Segmentation: Segmentation involves classifying each pixel of the image belonging to things. In this case, every pixel is labelled as belonging to a class, but no different instances of the same class are recognized. The polygon that delineates the object is called "segmentation mask".
- Semantic Segmentation: Similar to Instance Segmentation, but every pixel has to be labelled, thus stuff detection is required.
- Panoptic Segmentation[38]: This recent (2018) task demands that each pixel of the image must have both a semantic label and an instance id, hence merging Semantic and Instance Segmentation.

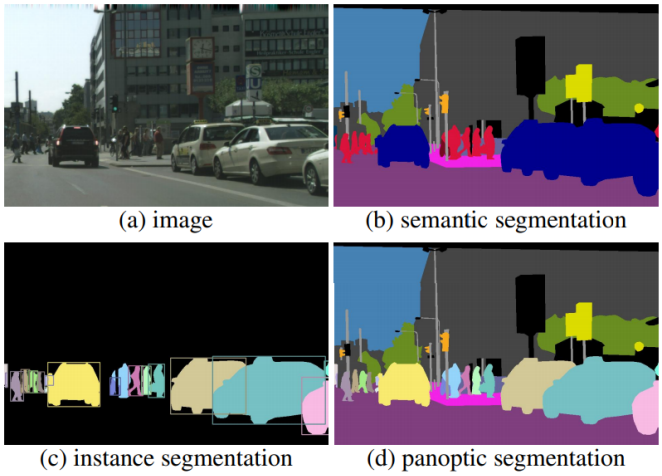


Figure 1.28: Comparison of different types of Image Segmentation. Taken from [38].

The addition of Markov Random Fields (MRF) and Conditional Random Fields (CRF) to segmentation models improves their performance[39]. There are ongoing efforts in the field advancing segmentation in 3D environments.

While most of the tasks named are usually solved with a CNN or variations, there are other tasks in Computer Vision that use different NNs. For example, Image Captioning is normally done with RNNs. This task consists in generating captions for images, which demands both language and image understanding.

Figure 1.29: Examples of Image Captioning. These captions have not been generated by a human. Taken from [91].

1.5 Training and Datasets

At each layer of the network, each node performs a computation. As seen in Section 1.3.1, the output of the neuron depends on the weights. Given a network, the process of optimising the weights so that the error rate is minimal is called "training".

It takes enormous amounts of data to train a NN. Sets of data are called "datasets" and the advent of Big Data and the ease of modern data collection have been a key reason for the recent explosion in Artificial Intelligence. NNs are trained on datasets, and they have reached current precision levels thanks to the amount of data available for training. For Computer Vision, a dataset consists of a large number of images that are labelled. In its most basic form, a label is simply a word associated to an image, denoting its class. More advanced labels include the position of ground truth bounding boxes locating objects, ID numbers, difficulty/occlusion indicators and more.

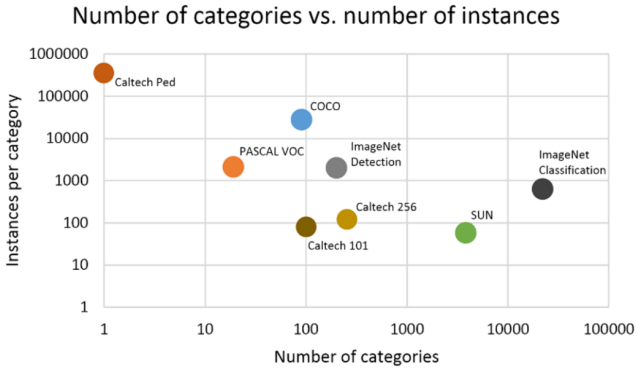


Figure 1.30: Sizes of some of the most common image datasets. Taken from [91].

The actual implementation of the training process varies depending on the nature of the algorithm to be trained (supervised vs unsupervised), but at their core both are minimising the error rate. Prior to that, the error rate must be quantified. It is called "loss function", and measures how different is the output of the network to the ground truth of the input. In the case of an image classifier, the ground truth for a picture of a cat is the class "cat", while the error rate would be the difference between the probability

produced by the network (cat:0.3) and the ground truth (cat:1). This would be a linear (identity) loss function.

1.5.1 Loss Function Minimization

There are several loss functions with slightly different properties. Some of the most common loss functions are:

Function	Equation
Cross Entropy	$-\sum_x p(x)\log q(x)$
L1	$\sum_i \lvert\hat{y}_i - y_i\rvert$
L2	$\sum_i \left(\hat{y}_i - y_i\right)^2$
Hinge	$\max(0, 1 - yw \cdot x)$

Figure 1.31: L1 full name is Mean Absolute Error, and L2 is Mean Squared Error. Although some loss functions have been traditionally used for classification and others for regression, this barrier may be diffuse[40] and more investigation is needed.

If the loss function is quadratic or exponential it will handle better outliers, or data far from the usual distributions that must be discarded. Behaviour near 0 or 1 is also important, as functions with greater slopes may be too volatile for precise fine tuning. The derivative is also important due to interaction with the optimization function. Loss functions also interact with the activation functions in the neurons. All these functions (activation, loss, optimization) are interrelated and must be chosen with care[41][42][43][44].

Before feeding the dataset to the model, it has to be partitioned. All datasets are too big to fit in the memory of any computer, so they are split into smaller pieces called "batches". Batches are fed to the model with initially randomized weights in the neurons. Then, the network processes the input and produces some output. Loss function of the output is computed, and is then passed to the optimization algorithm. Lots of algorithms exists for this purpose, the most common being based on gradient descent (Adagrad, Adadelta, Adam, RMSprop)[45]. The optimizer will return a set of weights that have a lower loss function value, and the weights are updated in a process called "backpropagation".

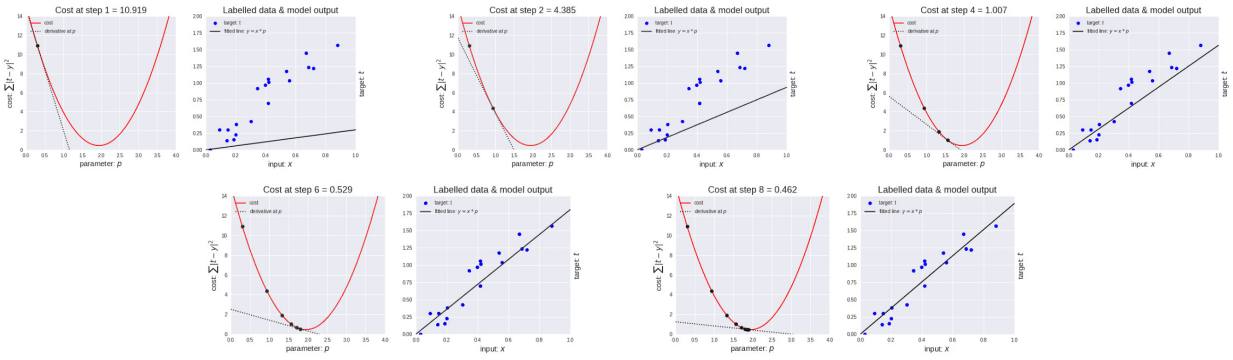


Figure 1.32: Optimizing a simple regression model with a single weight, the slope. In each of the images, the left plot is the loss function. Taken from [46].

Each time that a batch is processed is called a "step" (note that a batch can be further subdivided), and each time the entire dataset is processed is called an "epoch". "Iterations" is the number of batches

that the dataset has been divided into. Therefore, a 10,000 images dataset sliced into batches of 500 images will require 20 iterations to complete an epoch. The training process is finished when the loss function is low enough (depends on the dataset and the model, but generally near or lower than 1), which typically requires more than one epoch.

1.5.2 Validation and Overfit

When training there is a trade-off between generalization and accuracy. If the training loss is not sufficiently low, precision will not be as high as desired. Nevertheless, there is greater danger in trying to optimize too much the loss, causing overfitting. This is the name given to the situation when the detector is very reliable and precise when used with its training dataset, but has lower than expected performance when used with a different dataset. It usually is explained by a lack of generalization ability by the detector, that increasingly relies on recognizing just the exact same disposition of pixels, without being able to extrapolate forms.

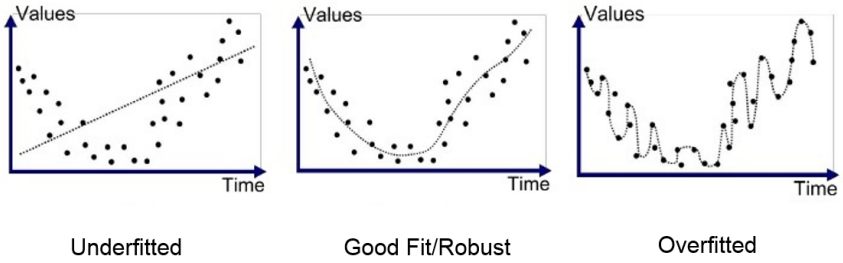


Figure 1.33: Examples of underfitting and overfitting. Taken from [47].

Underfitting is related to having high "bias" (error modelling the training data) and overffiting as high "variance" (not generalizing enough), as illustrated in Figure 1.34.

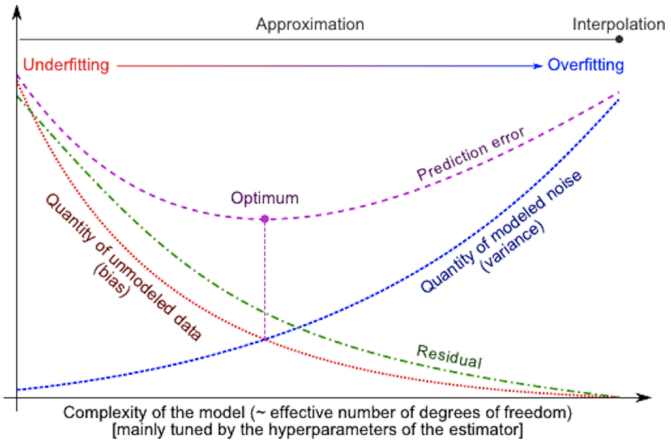


Figure 1.34: Plot describing bias and variance effect on prediction error. Taken from [48].

In order to avoid overfitting, a dataset used for training should be divided in two parts: training and validation (data augmentation techniques and dropout can also be used to fight overfitting). The training part is, as expected, images fed into the network and used for optimizing the loss function. The validation set, however, is not used in the optimization. It is used for assessing the detector's performance when exposed to images not used for the optimization and, therefore, "not seen" hitherto by the detector. Thus, if the training set loss keeps descending, but the validation set loss starts to rise, overfitting is happening. It is then best to either stop the training or tweak hyperparameters.

In addition, the validation set is used for tweaking the model's hyperparameters. These are parameters that are not optimized in the training process, because they do not belong to the network in the strict sense that a weight constant does. Instead, these parameters control other characteristics, such as network configuration (number of layers, architecture, resolution).

When the training process is finished, it is desirable to check the model's performance to assess precision improvements. But it should not be tested with the training dataset, because this result holds no real value. Since the model has been optimized to detect the ground truth in that dataset, the precision number obtained will just show how good or overfitted is the model to that dataset. Therefore, it is necessary to have another dataset, the test dataset, that must have not been used in the training process. This set can be a subset of the original dataset or an entirely new dataset. In the former case, if the dataset is small it may be challenging to split it into three parts (training, validation and test) while retaining a number of images sufficient to successfully train the model, since a common validation set size is 20 to 30% of the original dataset (training plus validation).

Chapter 2

State of the Art

2.1 Detection

2.1.1 Deep Learning Recent History

In 2012, a submission to the ImageNet ILSVRC Challenge (see Section 2.4.2) won the challenge by a margin of over 10 points with the second best participant. This breakthrough was made possible by the introduction of a deep (by 2012 standards) CNN using 2 GPUs. This model pioneered data augmentation techniques, ReLU activation functions and using parallel computation on two GUs. This model started a revolution in the Computer Vision field and has been cited more than 24,000 times.

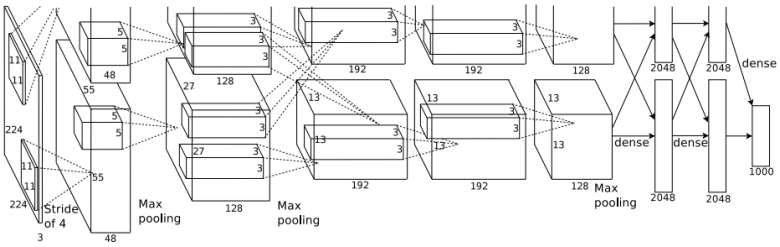


Figure 2.1: AlexNet CNN model. It was actually split in two networks for each GPU. Modern libraries allow for a single network to be trained with several GPUs without splitting them. Taken from [53].

The winner of the 2014 ILSVRC competition was Google's GoogleNet[54]. It achieved an error rate of 6.67%, with the human error for this dataset being around 5% [55]. This network incorporated Inception modules, batch normalization, image distortions and RMSprop as optimization algorithm. The network had only 4 million parameters, as opposed to AlexNet's 60 million.

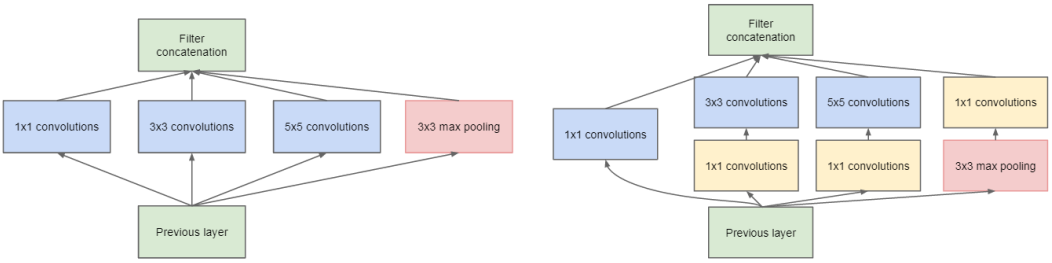


Figure 2.2: GoogleNet's Inception module (left) with dimensionality reduction (right). Taken from [57].

Inception modules combine convolution filters of 1x1, 3x3 and 5x5 with max pooling, passing the output of all these filters to the next layer. The problem with this design is the computational cost, as 5x5 filters are expensive. This was solved using 1x1 filters for dimensionality reduction. For example, using 20 1x1 filters, an input of size 64x64x100 (with 100 feature maps) can be compressed down to 64x64x20. A 2017 model inspired by Inception, Xception[58], used Inception modules to build a network that look into the three RGB channels separately, arguing that learning a 2D+1D correlation is easier to learn than a 3D mapping[57]. It has superior performance in larger datasets than Inception with greater computational efficiency.

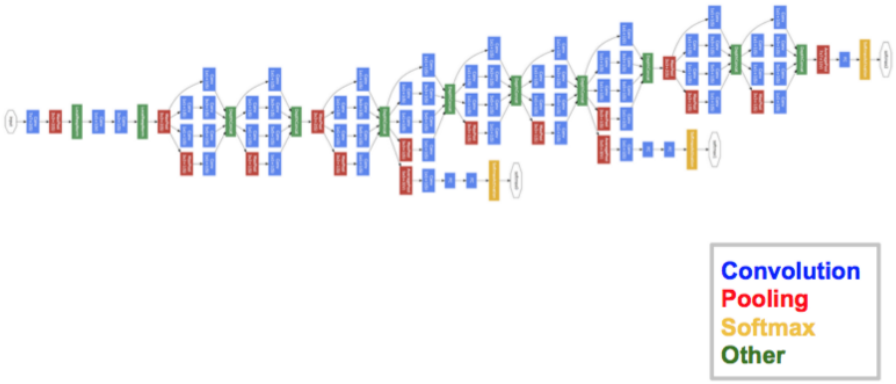


Figure 2.3: GoogleNet network. Taken from [54].

The next remarkable network was 2015 ILSVRC winner ResNet[35], which surpassed human level performance and was the first network to introduce "residual learning" and "ultra deep" networks, with 152 layers.

2.1.2 Detection Paradigms

Current object detection state of the art is dominated by Region of Interest (two-stage) and Single-Shot (one-stage) detectors. Both will be reviewed in this Section, along with the intuitive paradigm of sliding window.

Sliding Window

This basic but outdated paradigms is based on sliding a "window" over the image and feeding the cropped imaged to a classifier, thus detecting an object in the locations where the window obtained a high score. Nevertheless, it has a basic flaw regarding object sizes. The window has a fixed size but objects may appear in varying sizes, requiring differently sized windows.

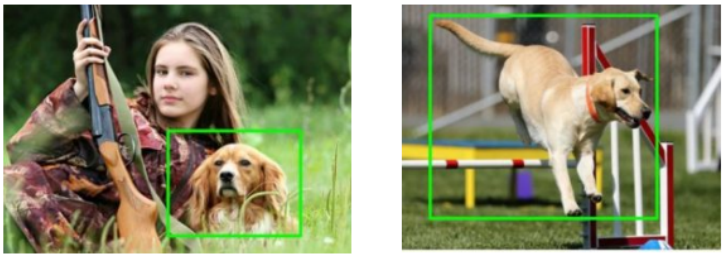


Figure 2.4: Objects appear in different sizes, making fixed sizes sliding windows problematic. Taken from [51].

The solution to this problem is resizing the image at multiple scales, so that at one scale the window will have the right size to detect the object. However, this method does not account for aspect ratio variation.

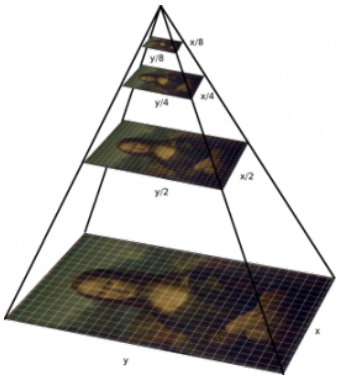


Figure 2.5: Resizing the image at different scales helps with the object size problem. Taken from [51].

A breakthrough in Computer Vision in 2005, the use of HOG (Histogram of Oriented Gradients) features by Navneet Dalal and Bill Triggs[52] allowed for successful detection of pedestrians using a Support Vector machine (SVM). See Section 2.2.1 for a definition of HOG.

Two-stage

Though replacing HOG based classifiers with CNN may sound like a good idea, running a CNN on each crop generated by the sliding window (with differently scaled versions of the input image) is computationally very expensive. To solve this, Two-stage detectors search for "Regions of Interest" (RoI), which are then passed to the CNN, reducing the number of potential bounding boxes processed to a few thousands.

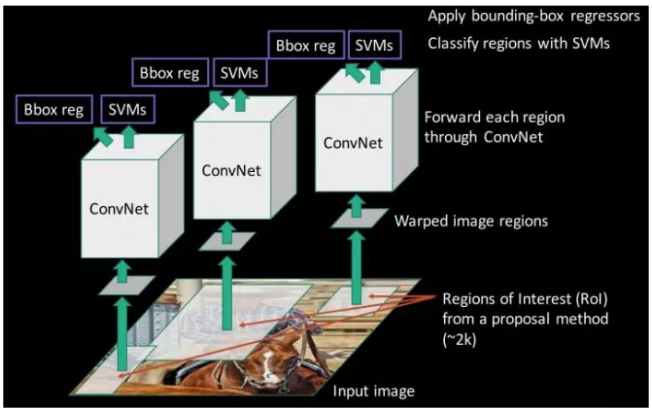


Figure 2.6: Basic structure of a two-stage detector. A region proposal algorithm signals potential object locations, and passes to the network only these regions, reducing the computational cost compared to processing the entire image multiple times as with sliding window methods. Taken from [51].

Instead of just passing a cropped patch to the CNN, the region proposal algorithm usually generates a feature map of the patch that is fed into the classifier. Development of RoI-based detectors is closely tied to the R-CNN model and superseding versions.

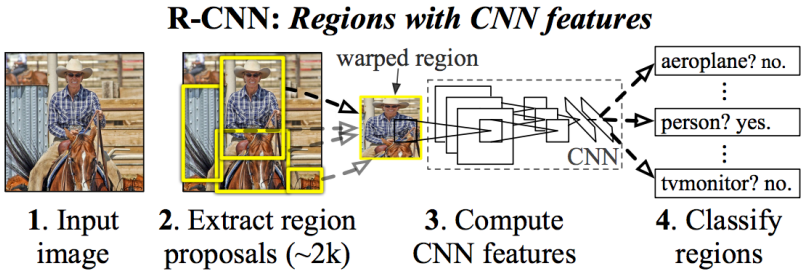


Figure 2.7: R-CNN architecture. Taken from [59].

R-CNN[59] was a highly successful two-stage model that pioneered ROI. Region proposal was made by the Selective Search algorithm, features were extracted with a CNN, but classification still used SVM. Training was very complex, having to train separately the feature generator, the classifier and the bounding box regression model[60]. Improvements to this model materialized in Fast R-CN[61].

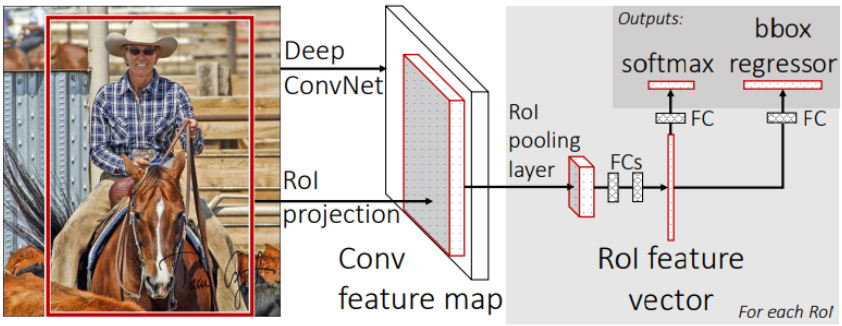


Figure 2.8: Fast R-CNN architecture. Taken from [61].

Fast R-CNN improved the training process by merging all the previously separate models into a single network. Computational load was softened by the use of RoI pooling. However, Selective Search algorithm remained a bottleneck. Efforts to solve this issue gave birth to Faster R-CNN[62], where Selective Search algorithm was replaced by a small CNN.

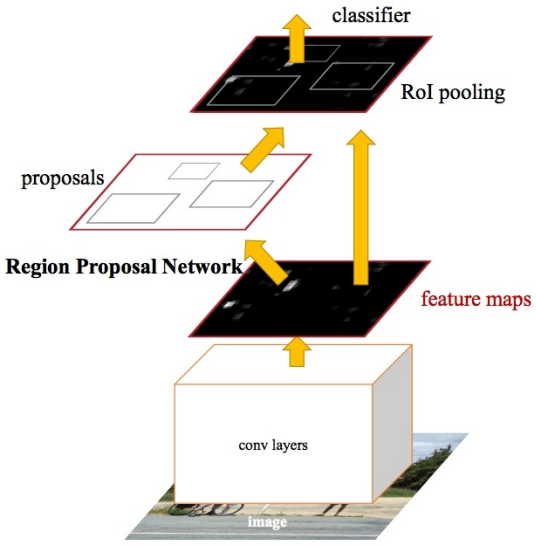
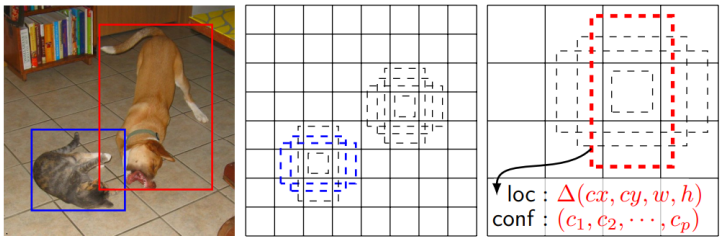


Figure 2.9: Faster R-CNN architecture. Taken from [62].

Faster R-CNN remains as the most accurate model in the Tensorflow suite[63] and is orders of magnitude faster than its predecessor, R-CNN. This architecture has inspired many other networks, such as its successor Region-based Fully Convolutional Net[64] (R-FCN) and its object segmentation derivative Mask R-CNN[65] (part of the Detectron AI software system owned by Facebook).

One-stage

One-stage detectors perform region proposal and classification simultaneously. Features maps of different scales are generated from the input, and for each location of a feature map, use convolutional filters to classify and predict the bounding box. The most famous detector of this type is SSD[67]. YOLO is the other most prominent example of this kind, and will be discussed in Section 3.2.



(a) Image with GT boxes (b) 8×8 feature map (c) 4×4 feature map

Figure 2.10: Feature maps of SSD detector. Taken from [67].

This architecture is challenging to train, since it generates lots of negative bounding boxes. Two techniques are used to mitigate this issue: non-maximum suppression groups together highly overlapping boxes into a single box, and hard negative mining uses negative examples with the highest training loss at each iteration of training, with a ratio of 3:1 negatives to positives.

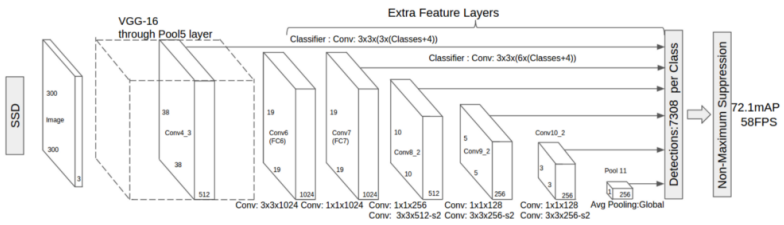


Figure 2.11: Architecture of SSD. Taken from [67].

An thorough comparison between two-stage and one-stage models is performed in [93], but the basic premise is that one-stage models are faster but less accurate. Attempts to bridge the gap in accuracy while retaining the speed have been made, such as RetinaNet[68].

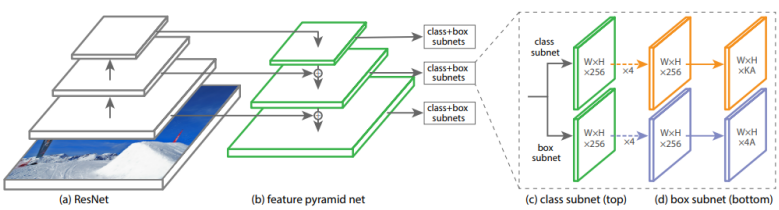
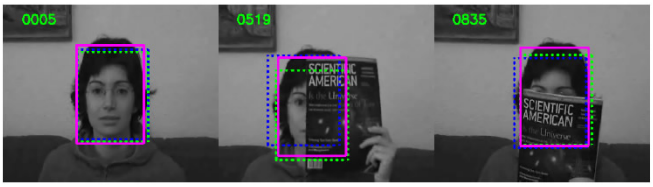


Figure 2.12: Architecture of RetinaNet. Taken from [68].

2.2 Tracking

The most basic definition of the problem of tracking is predicting the location of an object at t given its position at $t-1$. Generally, the whole past object location history is available, and an offline tracker will have access to the whole video sequence, both past and future. This distinction will be discussed in Chapter 2.2.2. Three fundamental problems make this problem non-trivial.



(a) Occluded Face 1

(c) Person

Figure 2.13: Occlusion happens when an object is behind another object and is therefore partially or completely hidden. Taken from [80].

The first of all is occlusion. When a target is occluded, information about its trajectory is lost. If the target modifies its trajectory, extrapolation from previous frames will fail to predict its position when the occlusion ends, thus terminating the tracking. Appearance changes and similarity is the second problem. Similar targets or changes in the appearance of a target may mislead the tracker and provoke a switch of assigned identities, or tracking failure. Furthermore, appearance changes could happen during occlusion, hampering any re-identification effort.

The third problem is tracking drift. Akin to guided missile GPS navigation, the probability of the tracker making a mistakes accumulates with time. As seen in Figures 2.14 and 2.15, drift deteriorates tracking performance significantly. Only very short sequences (under 20 seconds) do not suffer importantly from drift.

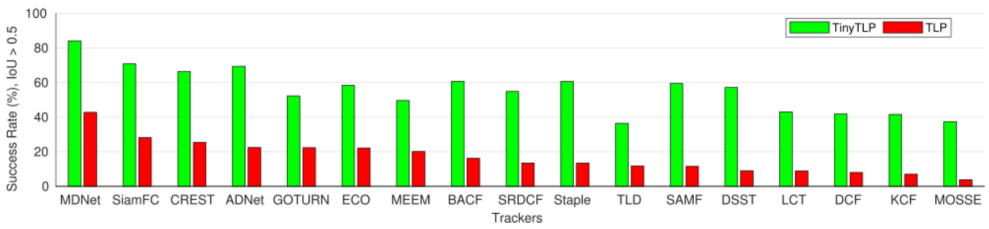


Figure 2.14: Tracker success rate (Intersection over union (IoU) of predicted and ground truth bounding boxes larger than a given threshold) of a number of state of the art trackers in the TLP (long term) and TinyTLP (short term) datasets. Taken from [118]. The TinyTLP dataset, consists of first 600 frames (20 sec) in each sequence of the TLP dataset to compare and highlight the challenges incurred due to long term tracking aspect.

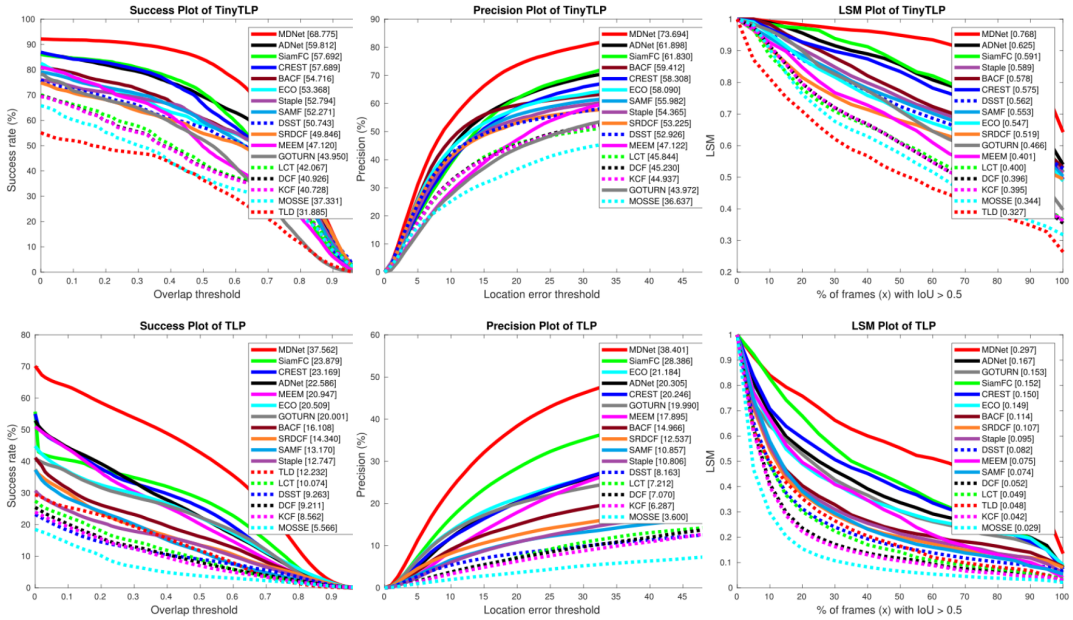


Figure 2.15: Tracker success, precision and LSM of a number of state of the art trackers in the TLP (long term) and TinyTLP (short term) datasets. Note that better metrics are achieved in short term tracking. Taken from [118].

2.2.1 Basic Tracking Techniques

Given the object location at $t-1$, there are three basic operations that can be done to locate it at t. Usually, a combination of them is employed in different order both in pre- or post-processing in order to achieve maximum precision.

Template Matching

Template Matching techniques are used for finding a template object in another image, called search image. Using simple image processing techniques such as Cross-Correlation or the Correlation Coefficient one can find the region of the search image that yields the greatest coefficient, and thus is the most similar to the template.

$$Cross\ Correlation = \sum_{x,y} Image_1(x,y) \times Image_2(x,y) \tag{2.1}$$

Figure 2.16: This expression of the Cross Correlation is easily distorted by brightness changes. This can be solved by normalizing each image with the average pixel value.

Another such method is sum of squared differences (SSD), which is one measure of match that based on pixel by pixel intensity differences between two images. It calculates the summation of squared for the product of pixels subtraction between two images [72].

Although basic in theory, this methods can be enhanced by other processing algorithms such as:

- Grayscale-based Matching: Inputs rotated versions of the template along with the original, allowing for rotation detection.
- Pyramid Processing: Downsampling the search image allows for a much faster processing and usually does not involve accuracy loss. After matching a region in the downsampled image, a

higher resolution version is used for better precision, until it is precisely located in the original search image.

- Edge-based Matching: As the shape of an object is defined by its edges, searching only around regions with similar edges significantly improves performance. In order to allow Pyramid Processing with this method, edge gradient is used instead of pixel color, because color resolution degrades with downsampling much faster than the gradient. Some edge detection algorithms include:
  - Gielis Curves
  - Hough Line and Circle Transforms [69]
  - Shape Descriptors [70]

The techniques reviewed in this section are successful at recognizing small, simple and predictable objects. Therefore, they have difficulties in detecting complex patterns. However, they still have interesting applications [71] where it would not be necessary to use a Neural Network, and progress is being made [73] in improving their robustness and precision.

As a side note, apart from using these algorithms in conjunction with Neural Networks to improve detection and tracking, they can be computed with Neural Networks [74] [75], an interesting development that increases accuracy.

Trajectory regression

Should information of past locations be treated simply as position and velocity data of a moving object, it can be processed with techniques from sensor analysis and control systems. One of the most used is the Kalman Filter. This algorithm combines a series of measurements that contain statistical noise and produce a probability estimation distribution for the next frame. A significant smoothing of the trajectory is achieved. Though an integral part of a number of trackers, it can not be used alone because it does not handle occlusion nor re-identification at all. A tracker using Kalman Filters is discussed in Section 3.3.

Feature Matching

A feature descriptor is a representation of an image or an image patch that simplifies the image by extracting useful information[77], typically converting an image of size width x height x 3 (channels) to a feature vector. Features can be abstracted as keypoints in an image such as corners, borders or shapes that describe the image. Instead of trying to match all pixels of an image as in Template Matching, only points that provide useful information are used, easing the computational cost. Extracting features from one image and comparing them with those of the target image will give a measure of similarity.

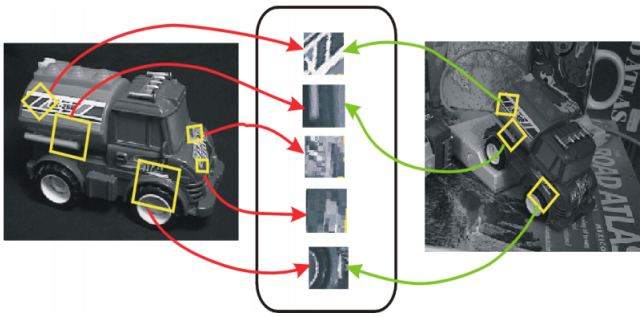


Figure 2.17: Feature descriptors. Taken from [77].

Three non Neural Networks techniques can be used:

- Color Histograms: Features can be extracted from the three color channels of images. This can be done by obtaining the histograms of various color channels and then comparing them.

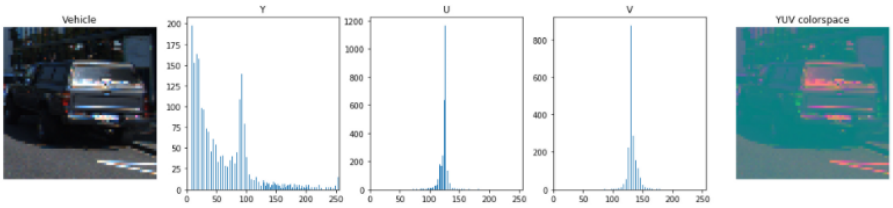


Figure 2.18: Color Histogram of a picture of a car. Taken from [78].

- Spatial Binning: Combining clusters of similar pixels into a single pixel reduces the amount of data and facilitates the analysis, though at the cost of loss of information.

Figure 2.19: Spatial Binning of a picture of a car. Taken from [78].

- HOG (Histogram of Oriented Gradients): The distribution (histogram) of directions of gradients is used as feature. Gradients (x and y derivatives) of an image are useful because the magnitude of gradients is large around edges and corners (regions of abrupt intensity changes) and small around flat areas[77].

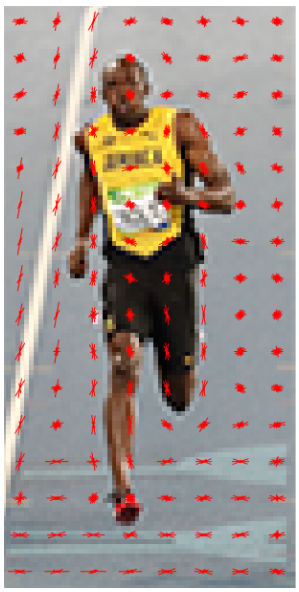


Figure 2.20: Visualization of the HOG descriptor of a photo of Usain Bolt. Taken from [79].

These are all traditional algorithms, but feature extraction can also be performed with Neural Network. A number of detectors and trackers incorporate small NNs (mostly CNN or Residual Networks) to extract features. A tracker using a NN to extract features is discussed in Section 3.3.

2.2.2 Tracker Classification

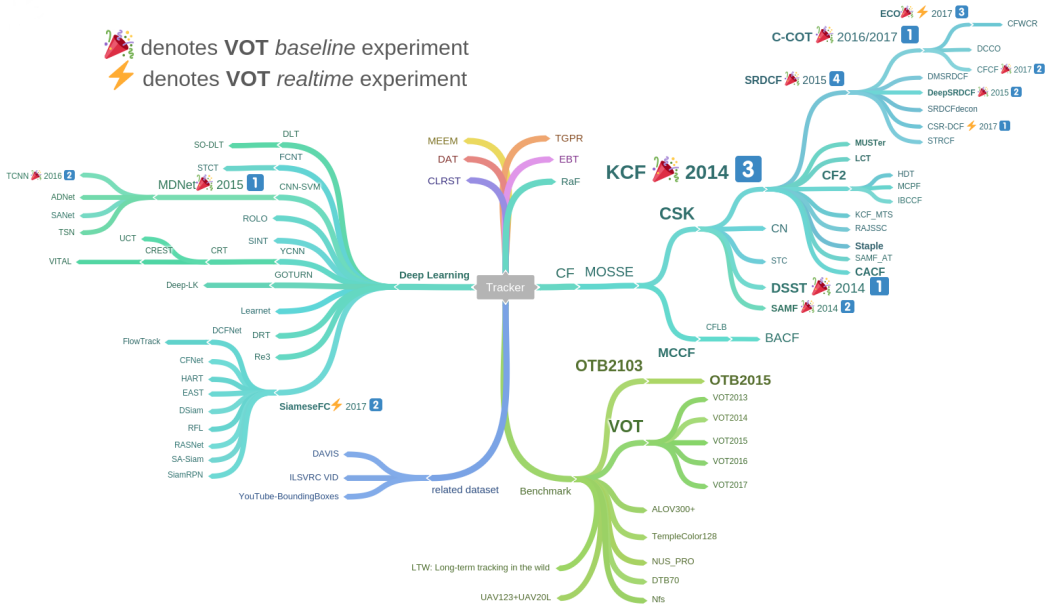


Figure 2.21: Recent tracker development branches. CF means Correlation Filters. Taken from [119].

There are lots of strategies and architectures for object tracking, and several models and algorithms can be used. Note that it is perfectly possible to build online trackers without Neural Networks that achieve similar results, at least until recently ([86][85][83][86][87]). Also, apart from CNN, RNN can be used [82] and so do GAN [81]. In this Section, a classification is offered to help distinguish the architectures and characteristics of trackers.

Online/Offline Tracking

The first distinction is whether the tracker operates offline or online. Offline means that the entirety of the tracking sequence is available, thus both past and future of current frame are taken into account when processing. On the other hand, online tracking means that only past and current frames are available, that is, the tracker is operating "live". Which one to choose lies in mission requirements and available computing power.

Offline tracker involves loading the whole sequence in memory, which may be impractical. Due to this constraint, even if there is no necessity to operate online it could be inevitable. However, achieving fast online tracking is difficult, as will be shown in Chapter 3. If the tracker operates at a slower speed than the video feed it is receiving, typically 25-30 FPS, then it is not usable and it is relegated to mission applications where an offline tracker would be used but there is not enough computing power.

Online/Offline Learning

A second classification is based on how the model was trained. Offline learning means that the tracker was trained on a dataset, and the resulting model is frozen and put to the task, not being affected by whatever it is tracking. This way, the model may fine-tuned to a specific kind of targets, and performance is better than their offline counterparts, but the tracker is lacks adaptability and is limited to the result of the offline training process.

Online learning, however, relies on the tracker being trained at execution. The model is updated at real time with the current appearance of the target, thus it learns and adapts to variations of the target (perspective, illumination). The promise of better tracking because of adaptability comes at the cost

of increased computing power required. Nonetheless, as the training process is fed what the tracker previously detected and there is no supervision, detection errors are propagated and accumulate in what is known as "tracking drift". There is no possibility of target recovery either.

Hybrid models exist, combining offline and online learning. The base model is trained offline, but either the last layers or some recurrent parameters are update on execution, allowing it greater adaptability while retaining offline learning performance.

Generic/Class Detection

A first approach at the tracking problem may consist of a combination of a detector, which provides approximate locations of the target most of the times (when not occluded, etc.), and a tracker, that accounts for past predictions and may even have feature extraction capabilities, trying to mitigate occlusion and re-targeting problem s (tracking-by-detection approach). This configuration is used, but there is room for innovation apart from the tracker. It is possible to train the detector so that, instead of recognizing only a handful of classes and thus be limited by its training, it recognizes objects in general.

Specifically, this models are trained to learn a generic relation between objects and motion. These models are not trained to recognize particular classes as "person" or "car", but instead generically extract features from the image at $t-1$. A region around the detection is defined where the object will potentially be at the next frame t. Different criteria exist for determining this region, the most intuitive being looking in an area around the detection at $t-1$, but more advanced algorithms may include linear regression of previous trajectory. This region of interest is cropped and processed, and the last layer outputs the predicted location of the object at t, which is then used as the basis for cropping and comparing the next frame $t+1$. Greater adaptability is achieved this way without the need for online learning, which results in greater performance. However, as no proper identification of the target is made, such models are prone to occlusion failures and identification switching.

Single/Multi Object

Although there is no theoretical obstacle in multi object tracking, that is, where more than one object is being tracked, practical considerations arise that make it difficult to generalize an existing single object tracker into a multi object one. For example, as generic object trackers are usually initialized with a bounding box provided by the user, work is needed to allow multiple objects being tracked and initialized at different times. As for class detectors, multiple objects means that at a given frame there exist several detections and identities that must me matched. Feature extraction and Decision theory play an important role in addressing this problem.

This is the architecture of trackers such as GOTURN [107] or Re^3 [108]. These Neural Networks are trained to detect generic motion, therefore no identity recognition is achieved besides frame-to-frame comparison. This makes occlusion of special concern because if the target is lost, there is no hope for recovery.

2.2.3 Tracker Fusion

An interesting option barely explored in the literature is that of using several trackers simultaneously and filtering the best results. Such model was studied in [126], and results show that this model performs as good as the best tracker, ignoring bad output from the other trackers. Though the performance analysis in the paper is lacking, the premise is promising and merits further investigation.

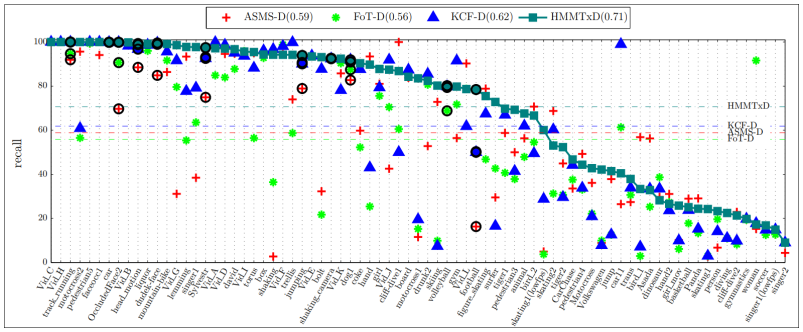


Figure 2.22: Per sequence analysis of the separate trackers combined with the detector (i.e. KCF-D, ASMS-D, FoT-D) and the proposed HMMTxD. The average recall is shown by the dashed lines. Black circles mark grayscale sequences. The sequences are ordered by HMMTxD performance. Taken from [126].

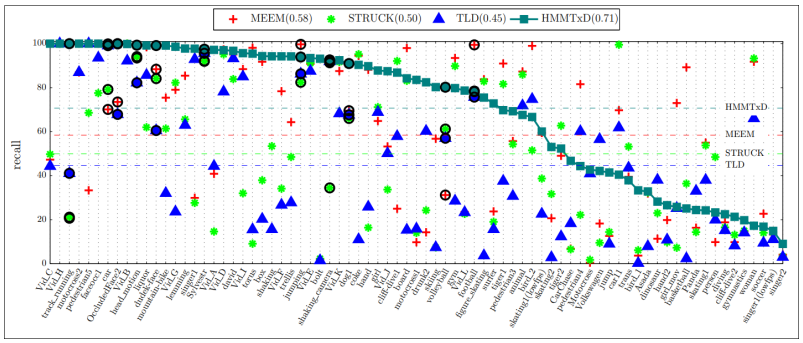


Figure 2.23: Evaluation of state-of-the-art trackers on the TV77 dataset in terms of recall, i.e. number of correctly tracked frames. The average recall is shown by the dashed lines. Black circles mark grayscale sequences. The sequences are ordered by HMMTxD performance. Taken from [126].

2.3 Detection Metrics

First of all, a quick reminder. A detector outputs bounding boxes but, as in object classification (and most detectors are fundamentally classifier), it also gives a confidence score (from 0 to 1) for each bounding box. It is exactly the same concept as in classification, with the score representing how sure the model is that the object belongs to the predicted class.

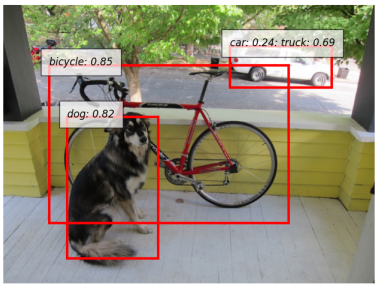


Figure 2.24: A detector outputs both a bounding box and a confidence score associated to each bounding box. Taken from [128].

Depending on how the detector is built, it will most probably output dozens or hundreds of bounding boxes for a single image. Nevertheless, most of them will have very low confidence scores, therefore

a threshold is set to retain only the meaningful ones. Detections below that confidence score threshold (typically ranging from 0.3-0.6) are discarded. This confidence threshold can be modified by the user depending on the desired output and the input characteristics. With this problem sorted, detection metrics can be defined.

The most basic concepts upon which any Computer Vision metric is built are those of True Positive, False Positive and False Negative.

- True Positive: Target exists and is detected.
- False Positive: Target does not exist but is detected.
- False Negative: Target exists but is not detected.

Whether a detection is considered a True Positive or a False Negative depends on which criterion is used. Two criteria are common in the field, centre distance and intersection over union (IOU). They are simple: centre distance is the euclidean distance of the centre of the ground truth bounding box and the detection bounding box, while IOU is the quotient of the overlap area of the two boxes and the union of the areas, as shown in Figure 2.25.

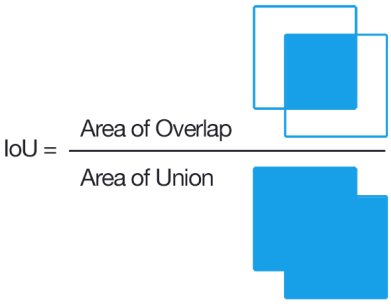


Figure 2.25: IOU concept explained graphically. Taken from [127].

Both are equally valid, given that the focus is on comparison rather than the actual metrics' significance. For this work IOU will be used, since it is more widespread. Figure 2.26 gives a quick grasp of usual values of this criterion. Typically, IOU = 0.5 is used as threshold.

Figure 2.26: IOU values shown graphically. Very low values indicate no detection, below 0.5 is a poor detection, above 0.5 is a good detection and near 1 is a perfect detection. Taken from [127].

With these concepts we can define more complex metrics:

- Precision is the proportion of all detections that are correct:

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive} \tag{2.2}$$

To put it in perspective, a Precision value nearing 1 means that whatever the model detects is almost always a correct prediction.

- Recall is the proportion of all ground truth bounding boxes that have been correctly detected:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{2.3}$$

To put it in perspective, a Recall value nearing 1 means that almost all ground truth objects will be positively detected by the model.

It is very important to note that there is usually an inverse relationship between precision and recall: lowering the confidence score threshold means more detections are considered, which will probably increase Recall, but Precision will decrease. If the confidence value is very low Recall may approach 1, but it would be meaningless, because the hundreds of almost random bounding boxes predicted just happen to correctly predict the ground truth out of pure luck (the sheer number of predictions causes that some of them will "predict" the ground truth, whichever its confidence score is). Also, remember that "correctly detected" and "True Positive" means that it satisfies the IOU threshold value. A True Positive detection may turn into a False Positive if the threshold value is set sufficiently high.

Therefore, if the confidence score threshold is varied, different pairs of Precision-Recall values are obtained, which are plotted in the Precision-Recall curve (Figure 2.27):

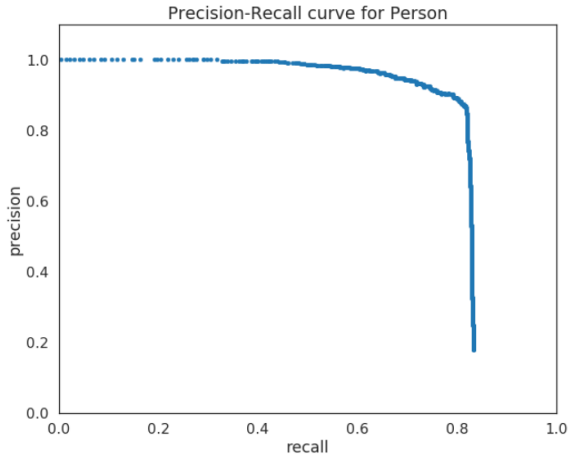


Figure 2.27: Example of a Precision-Recall curve. Taken from [129].

Finally, the Average Precision (AP) can be calculated. Its definition is the average of 11 values of Precision at the points where $Recall_i = \{0,\ 0.1,\ 0.2, \ldots\ 0.9,\ 1\}$.

$$AP = \frac{1}{11} \sum_{Recall_i} Precision(Recall_i) \tag{2.4}$$

This definition is explained by its creators:

> The intention in interpolating the precision/recall curve in this way is to reduce the impact of the "wiggles" in the precision/recall curve, caused by small variations in the ranking of examples. It should be noted that to obtain a high score, a method must have precision at all levels of recall-this penalises methods which retrieve only a subset of examples with high precision (e.g. side views of cars).
>
> The Pascal Visual Object Classes (VOC) Challenge [122]

However, this definition was changed in 2010. The latest AP definition is computed calculating the area under the Precision-Recall curve with Precision monotonically decreasing, by setting the Precision for recall r to the maximum Precision obtained for any Recall $r' \geq r$ [123]. A graphical example is given in Figure 2.28. This modification is enacted in order to address small variations and inconsistencies in the model's output.

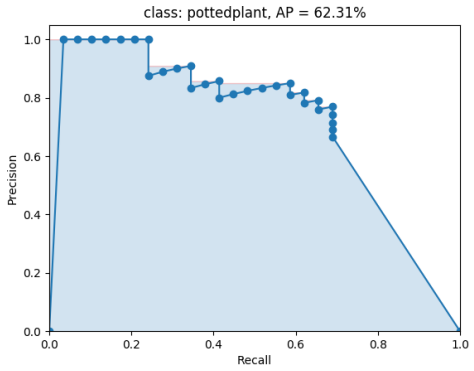


Figure 2.28: Precision-Recall curve with Precision monotonically decreasing. Red line indicates that instead of the actual Precision value, the higher Precision value in the red line will be used for the computation. Taken from [131].

This definition is the one used in the PASCAL VOC Challenge latest edition. This AP value applies only for one class. If the values of all classes are averaged the mean Average Precision is obtained, mAP. As only the class "person" will be used, mAP and AP are the same, but mAP will be used since it is the most common term.

However, the COCO challenge uses a slightly different definition. The Precision-Recall curve was obtained by varying the confidence score threshold (which deletes all detections below the threshold) but leaving constant the IOU threshold (which differentiates a good detection from a bad one). By varying also the IOU threshold a number of curves are obtained, as shown in Figure 2.29.

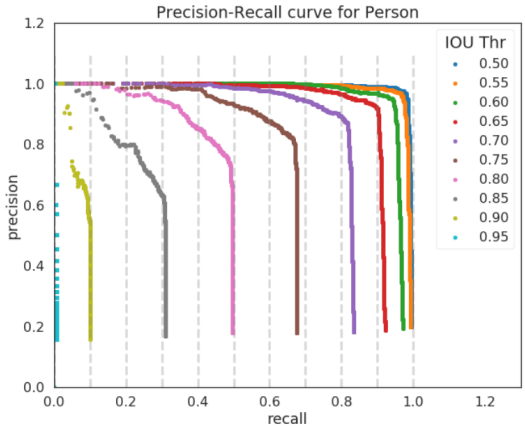


Figure 2.29: Precision-Recall curves with different IOU thresholds. Taken from [129].

The mAP value is obtained by averaging all classes and all IOU thresholds (a range of IOU thresholds from 0.5 to 0.95 in steps of 0.05 are used in the COCO Challenge). Averaging over IOU thresholds rather than only considering one generous threshold of IOU equal or greater 0.5 tends to reward models that are better at precise localization [129].

As can be gathered from this section, even a single, clear metric can be calculated slightly differently by different parties. Comparing metrics between different authors should thus be done with caution, and

any metric absolute value is of little importance without a precise explanation of how it was computed. The intended use of this metrics is to compare results in the same dataset and, if possible, computed by the same author.

2.4 Detection Challenges and Benchmarks

2.4.1 PASCAL VOC

One of the earliest datasets, and perhaps the most important among them (with only one precursor of comparable size, LabelMe[88], published in 2005). The size of the PASCAL VOC[122] dataset increased throughout the years, starting at 9,963 images containing 24,640 annotated objects in 2007 and 11,530 images containing 27,450 ROI (Region Of Interest) annotated objects and 6,929 segmentations in 2012, the last edition of the challenge. It was the biggest, fully annotated Computer Vision dataset by the time of its publication, with 20 classes.

PASCAL VOC adopted in 2007 the Average Precision (AP or mAP) metric, now the most popular metric for assessing performance of object detectors, and its annotation format is still one of the most used today. Both were used in this project and are discussed in depth (mAP in Section 2.3 and annotation format in Appendix A.1).

2.4.2 ImageNet

ImageNet[90] represented a great advance in dataset size. With over 14 million images, and more than 1 million images annotated (with a thousand classes). The ImageNet Large Scale Visual Recognition Challenge (ILSVRC, 2010-2017) saw the breakthrough of AlexNet in 2012 (Figure 2.30).

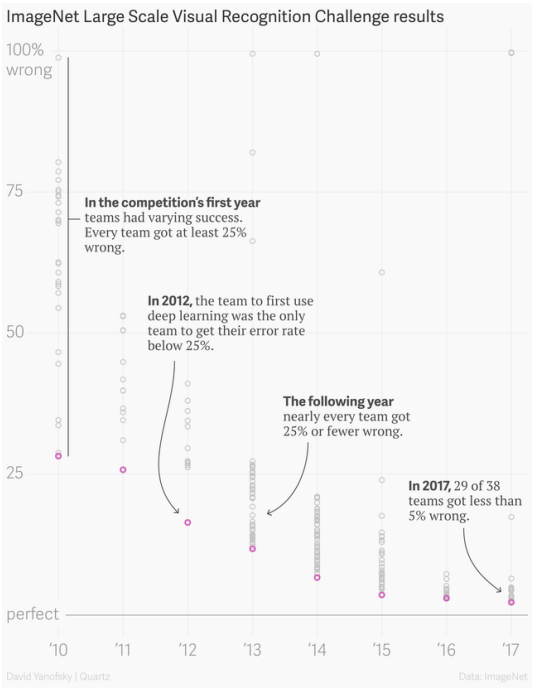


Figure 2.30: ILSVRC results between 2010 and 2017. AlexNet won the 2012 challenge. Taken from [91].

The herculean task of annotating the dataset could not be performed by computers at the time nor by a team of annotators (it would take some 90 years). Instead, Amazon Mechanical Turk was used. This online platform allows users to perform rather mundane tasks that computers are unable to do. Workers

(Turkers) receive a payment by the employer. Some tasks common are, for example, writing product descriptions, identifying performers on music CDs, social science experiments, artistic and educational research, and even missing persons searches.

2.4.3 MS COCO Dataset

COCO is perhaps the reference dataset at this moment. With over 330K images and 1.5 million object instances of 80 object categories (and 91 stuff categories), it is one of the largest datasets.

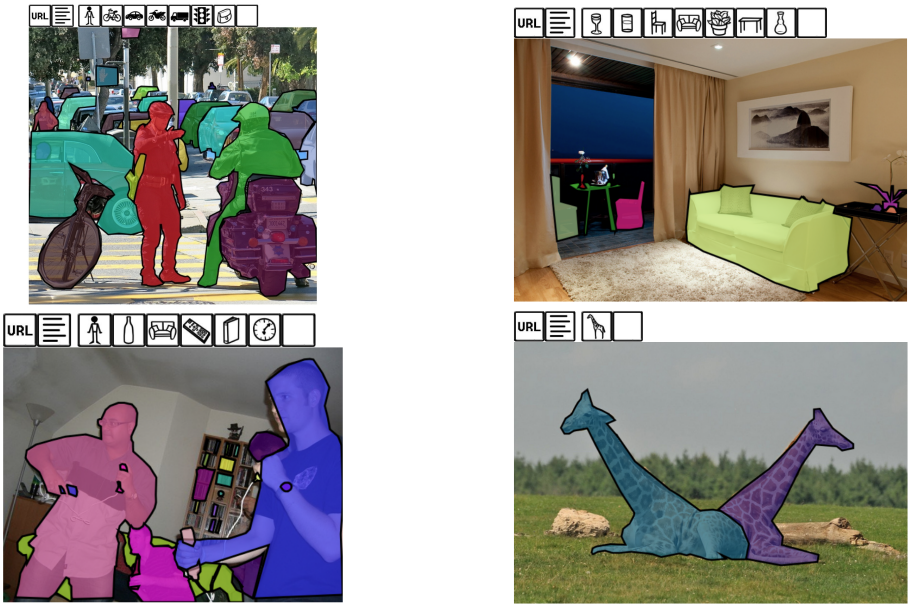


Figure 2.31: Example images from the COCO dataset. Taken from [89].

Several challenges are offered by COCO: detection (both bounding box and instance segmentation), keypoints, stuff (semantic segmentation) and panoptic segmentation.

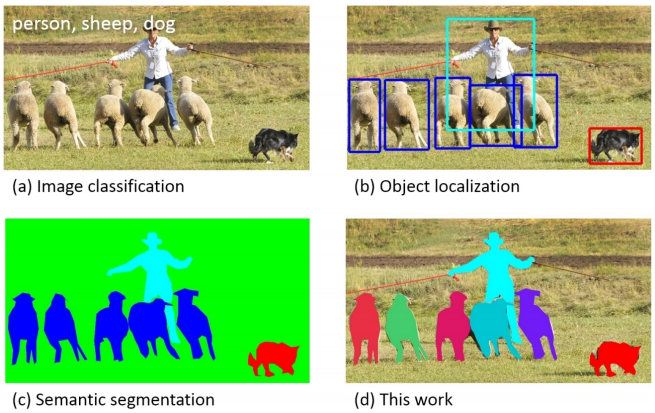


Figure 2.32: COCO dataset focuses on segmenting individual object instances. Taken from [89].

2.5 Tracking Metrics

Name	Description
MOTA	Multiple object tracker accuracy.
MOTP	Multiple object tracker precision.
IDF1	ID measures: global min-cost F1 score.
MT	Mostly Tracked: Number of objects tracked for at least 80 percent of lifespan.
PT	Partially Tracked: Number of objects tracked between 20 and 80 percent of lifespan.
ML	Mostly Lost: Number of objects tracked less than 20 percent of lifespan.
FP	False Positives.
FN	False Negatives.
IDS	ID Switch: Number of track identity switches.
FM	Track Fragmentations: Number of times a ground truth trajectory is interrupted (untracked).
Recall	Number of detections over number of objects.
Precision	Number of detected objects over sum of detected and false positives.

Figure 2.33: Measures from the MOT-CLEAR metrics and ID metrics. Metrics where lower is better: ML, FP, FN, IDS, FM. Higher is better: MOTA, MOTP, IDF1, MT, PT, Recall, Precision.

2.5.1 Fragmentation and ID Switch

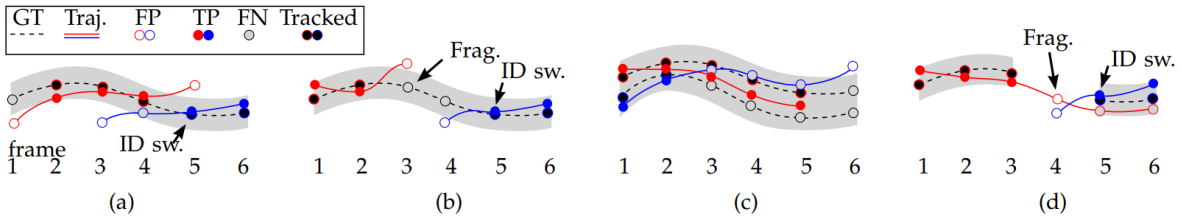


Figure 2.34: Illustration of fragmentation and ID switch concepts. Taken from [113].

A visual explanation of the concepts of fragmentation and ID switch is shown in Figure 2.34. An ID switch happens when the ground truth target that is being tracked by a certain ID trajectory starts to be most accurately tracked by another ID (case a). Fragmentation occurs when a ground truth target stops being tracked, and it is not re-targetted. If re-targetting happens after the fragmentation with a different ID, an ID switch is also counted (case b). Evaluating tracking is sometimes hard, as in case c, which shows two mostly correct tracking trajectories, but with poor assignments to ground-truth targets. Although this tracking would be considered acceptable, False Negatives and False Positives will be reported by the evaluation software. Lastly, case d shows that interrupted ground truth trajectories will cause fragmentations, and may also cause ID switches.

As will we mentioned several times in this work, metrics should be put in context and considered with a bit of scepticism. For example, a poor tracker that is not able to follow the target during a significant part of its trajectory will have poor Recall, but may show low (good) fragmentation and ID switch numbers, and good Precision, potentially misleading an observer's opinion. In order to mitigate this, fragmentations and ID switches should be given relative to number of detections or Recall.

2.5.2 Mostly and Partially Tracked, Mostly Lost

These three metrics offer more of a quick, qualitative estimation of tracking, without obscure metrics and the subtleties in their interpretation. These are Mostly Tracked (MT), Partially Tracked (PT) and Mostly Lost (ML). As defined in [114]:

- Mostly Tracked: A target is mostly tracked if it is successfully tracked for at least 80% of its life span. Note that it is irrelevant for this measure whether the ID remains the same throughout the track.
- Mostly Lost: If a track is only recovered for less than 20% of its total length, it is said to be mostly lost (ML).
- Partially Tracked: All other tracks are partially tracked.

These metrics should be reported as a ratio to the total number of ground truth trajectories.

2.5.3 MOTA and MOTP

MOTA and MOTP are part of the CLEAR MOT[113][114] metrics, perhaps the most widely used metrics to evaluate performance of trackers. The MOTA (Multiple Object Tracking Accuracy) is . The main reason for this is its expressiveness as it combines three sources of errors (FN, FP and IDS).

$$MOTA = 1 - \frac{\sum_t \left(FN_t + FP_t + IDS_t\right)}{\sum_t GT_t} \qquad MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}$$

Figure 2.35: MOTA and MOTP definitions. . t is the frame index, GT the number of ground truth objects, c_t denotes the number of matches in frame t, and $d_{t,i}$ is the bounding box overlap of target i with its assigned ground truth object.

The Multiple Object Tracking Precision is the average dissimilarity between all true positives and their corresponding ground truth targets. MOTP thereby gives the average overlap between all correctly matched hypotheses and their respective objects. It is important to point out that MOTP is a measure of localization precision, (although not exactly Precision as used in object detection). As it provides little information about the actual performance of the tracker it will not be used in this work, in favour of Precision.

2.5.4 Tracking Paradigms and IDF1

Measuring tracking performance with a single measure of performance is almost impossible, given the complexity of tracking and the different needs of different applications of tracking. Consider the following example and two possible evaluation paradigms.

Figure 2.36: Three trackers obtain these results when following a single target. Taken from [115].

Depending on the frequency or length of these confusions, two evaluation paradigms stand out.

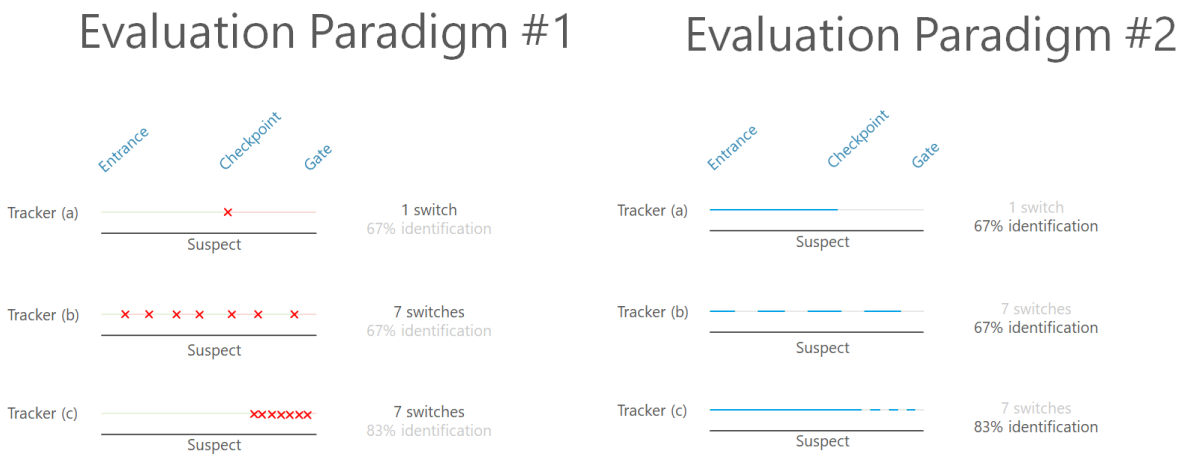


Figure 2.37: Illustration of fragmentation and ID switch concepts. Taken from [115].

The first evaluation paradigm examines how often a target is lost or reacquired. It measures errors through identity switches. According to this paradigm, tracker a is the best choice (1 switch) and b and c are equally worse (7 switches). The second paradigm instead evaluates how often a target is correctly identified, regardless how often it is lost or reacquired. According to this paradigm, tracker c is the best with 83% identification recall/precision and a and b are equally worse with 67% identification recall/precision.

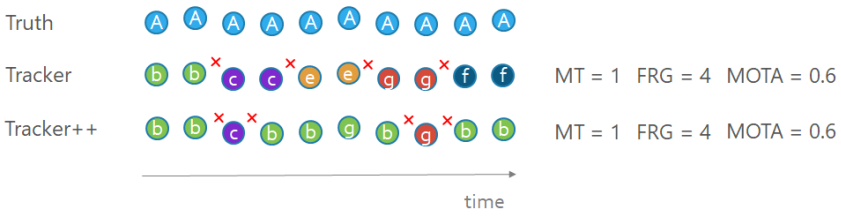


Figure 2.38: Two trackers obtain these results when following a single target. Note that when the tracker re-identifies the target it is also counted as a fragmentation. Taken from [115].

The two trackers in Figure 2.38 obtain the same MOTA score, the same MT score of 1 and the same fragmentations. However, intuitively, one tracker is better than the other. These examples illustrate the difficulty of this task, and how MOTA can be misleading. IDF1 metric is a better choice for evaluating tracking performance.

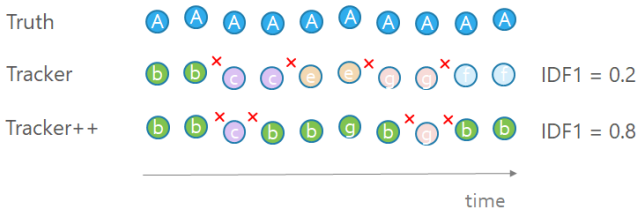


Figure 2.39: IDF1 depicts more accurately tracking performance. IDF1 definition is much more complex than MOTA and can be found here[116]. Taken from [115].

2.6 Tracking Challenges and Benchmarks

International challenges and conferences are useful ways of keeping up to date with the state of the art and discovering good trackers. Below is a quick overview of the two most relevant to the object tracking field.

2.6.1 VOT Challenge

The objective of the VOT Challenge, more specifically the 2017 challenge, is as defined by its creators:

> Like VOT2013, VOT2014, VOT2015 and VOT2016, the VOT2017 challenge considers single-camera, single-target, model-free, causal trackers, applied to short-term tracking. The *model-free* property means that the only training information provided is the bounding box in the first frame. The *short-term* tracking means that trackers are assumed not to be capable of performing successful re-detection after the target is lost and they are therefore reset after such event. *Causality* requires that the tracker does not use any future frames, or frames prior to re-initialization, to infer the object position in the current frame.
>
> The Visual Object Tracking VOT2017 Challenge Results [111]

VOT goals do not exactly align with the objective of this work, that pursues long-term tracking, but valuable information can be extracted from its methodology and results.

Methodology

The primary measures of the VOT challenge are accuracy, robustness and expected average overlap (EAO). As trackers are not expected to re-detect lost targets due to the *short-term* component of this challenge, when zero overlap between the bounding box and the ground truth occurs it is considered a failure, ant the tracker is re-initialized five frames after the failure. Ten frames after re-initialization are ignored in the accuracy measure to avoid bias due to resets.

- Accuracy is the average overlap between the predicted and ground truth bounding boxes during successful tracking periods.
- Robustness measures how many times the tracker loses the target (fails) during tracking.
- Expected average overlap (EAO), is an estimator of the average overlap a tracker is expected to attain on a large collection of short-term sequences with the same visual properties as the given dataset.

This is a quite simple system that, while enough for single object tracking purposes, does not capture all the complexities involved in tracking multiple objects. A more thorough system is employed in the MOT challenge and will be discussed in Subsection 2.6.2.

Results

VOT offers an in-depth study and ranking of dozens of trackers, but extracting general conclusions is the focus in this section. For example, the incredibly rapid advance of the Object Tracking field can be seen in Figure 2.40. 35% of the trackers submitted to VOT17 are already better (at least in this metric) than the average state of the art tracker published in 2016 and 2017, and trackers considered as references five years ago (such as MIL [85]) or even recently (KCF [86]) are situated at the lower quarter of the graph.

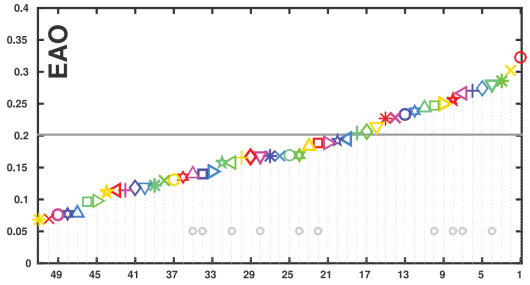


Figure 2.40: Tracker EAO (Expected Average Overlap) in VOT17, sorted by ranking. Grey line denotes the average performance of ten state-of-the-art trackers published in 2016 and 2017 at major Computer Vision events. These trackers are represented as grey circles in the bottom of the figure. Taken from [111].

Relevant to this work is the VOT realtime experiment. The VOT baseline experiment does not impose any speed requirements on the tracks, but the realtime experiment that made its debut in 2017 does. It expects trackers to operate in realtime (25 fps or Hz) and if a bounding box is not provided for the current frame, the previous bounding box is taken as output. The effects in both accuracy and robustness can be seen in Figure 2.41. Trackers that do not operate in or above real time see their accuracy and robustness hurt significantly, while fast trackers retain their precision.

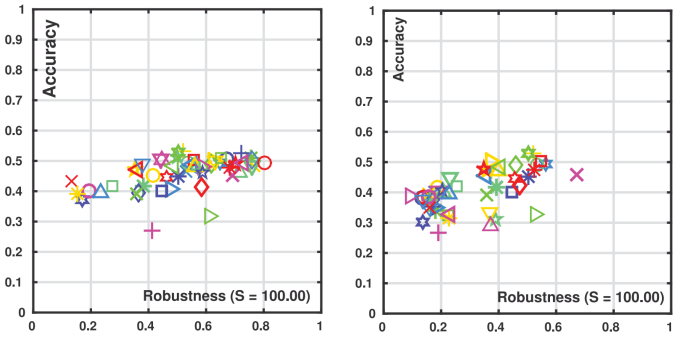


Figure 2.41: Accuracy and robustness comparison between the VOT17 baseline (left) and realtime (right) experiment. Taken from [111].

2.6.2 MOT Challenge

MOT (Multiple Object Tracking) Benchmark is a dataset and challenge that focuses on long-term, multiple object tracking. It is associated with the previously discussed CLEAR MOT metrics.

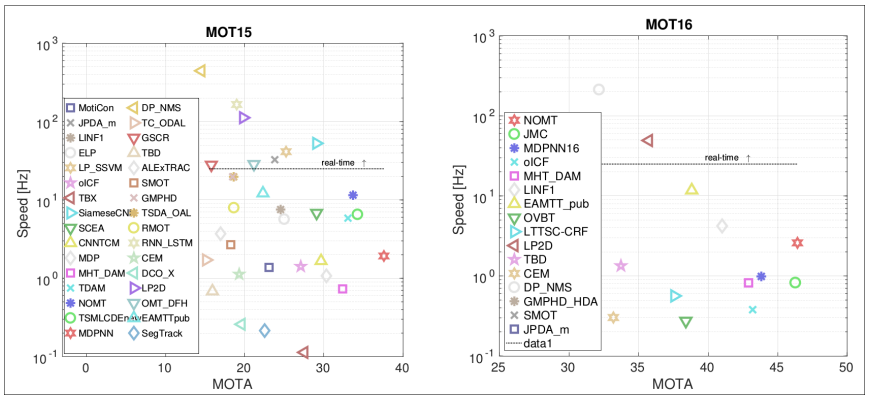


Figure 2.42: Tracker speed and performance measured in MOTA for the MOT Challenge 2015 and 2016. Taken from [117]. It is difficult to find trackers capable of operating at real time.

Goals of this challenge are close to the objectives of this work, and its metrics and results such as Figure 2.42 are used throughout this thesis. It should be the reference challenge to evaluate new trackers intended to perform long-term tracking.

Chapter 3

Algorithm

3.1 Algorithm Election: Practical considerations

Elaborating from scratch a Neural Network detector and a tracker is a task that requires substantial experience and academic formation, as well as an amount of time not available in the time span of a semester. Hence, being open-source was the first requirement for any existing Artificial Intelligence considered for this project. This cuts significantly the number of potential trackers available.

3.1.1 Hardware and Frame Rate

First of all, the model used for detection was limited by available hardware specifications at the workplace. While a more capable computer, equipped with a 5.1 CUDA NVIDIA GTX 1080, was used for training and testing, most of the initial work was done on a less capable machine, so memory limitations were a hard limit. In addition, at initial stages of the project it was considered delivering this software to a UAV for on-board computing. Using a Raspberry Pi was considered but quickly dismissed, although striving for a lightweight model remained part of the philosophy of this project.

Apart from the memory limitations that restricted which model was able to be loaded, the available ones had to perform faster than live video streaming at 25-30 FPS. It is difficult to achieve fast online tracking (see Figure 3.1). This greatly restricts the number of potential trackers, and in conjunction with the open-source requisite, makes it very non-trivial. Therefore, it was mandatory to put emphasis on performance at the cost of precision, since it is typically a tradeoff. Also, note the difference in performance between CPU and GPU for the same trackers in Figure 3.1.

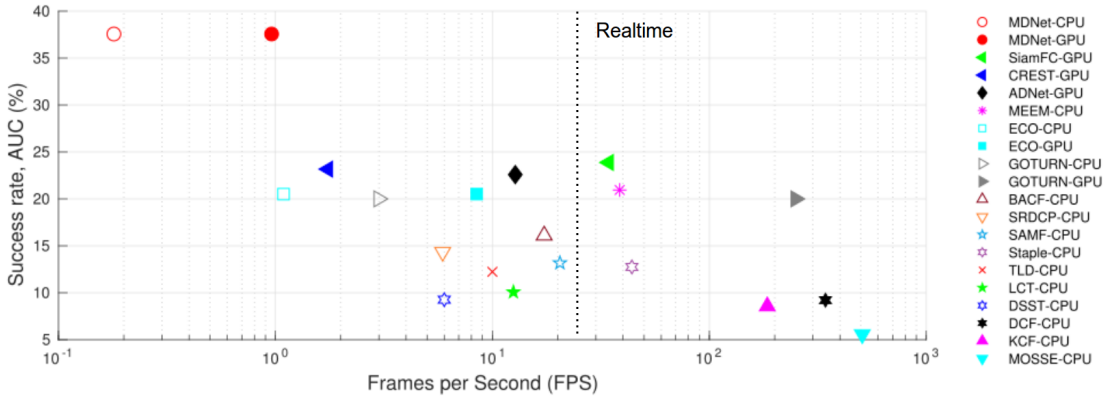


Figure 3.1: Tracker frame rate and success rate (Intersection over union (IoU) of predicted and ground truth bounding boxes larger than a given threshold) of a number of trackers. Taken from [118]. It is difficult to find trackers capable of operating at real time.

3.1.2 Programming Language

A number of programming languages are currently used, mainly Python, C++ and MATLAB, with Tensorflow and Caffe as the most used frameworks. Although this does not represent a hard limit such as performance, some considerations favor certain configurations above others.

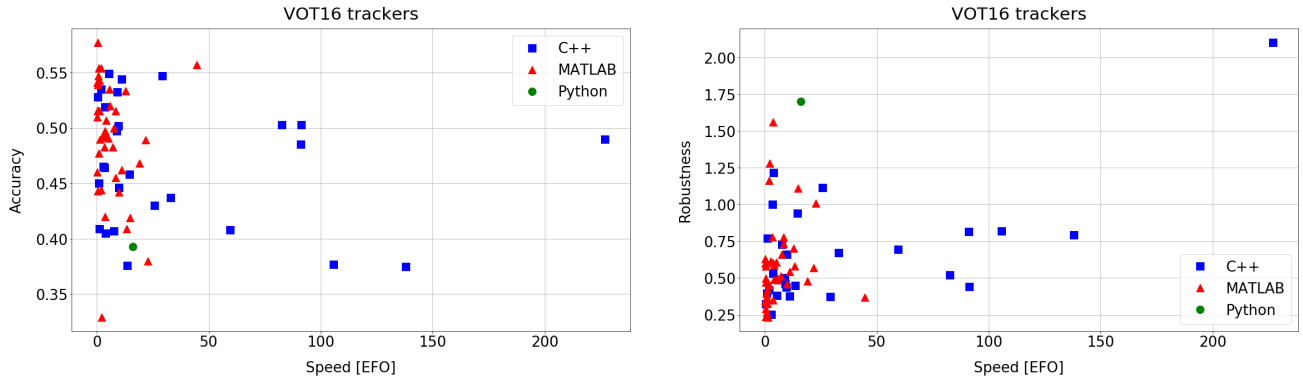


Figure 3.2: Performance of VOT16 of trackers with different languages of implementation. Data taken from [110].

Taking as reference the results in the VOT16 challenge [110] represented in Figure 3.2, it is observed that there is no clear advantage in using C++ above MATLAB, although trackers implemented in the latter seem not able to pass a certain speed threshold which some C++ based trackers can. Barely any trackers are implemented in Python, so little can be said of its performance.

Even though performance-wise C++ may be superior, optimization is beyond the scope of this work, so Python's readability was deemed more important. MATLAB was discarded because of a hardware-related limit. Hard drive space was limited and MATLAB installation is considerably large. Space had to be reserved for the selected Dataset, which typically occupy from a few to tens of gigabytes. Moreover, as integration with drone control software is an objective, MATLAB does not offer integration as satisfactory as Python or C++. Regarding the framework, Tensorflow was preferable because of the larger community and documentation available, as well as for the simpler installation.

3.1.3 Open Source and Multi Object

Few of the trackers submitted to challenges such as VOT or MOT are open-source, and only a subset of them have made their training code available. This is maybe the major restriction to face when selecting a tracker. Furthermore, some configurations are more desirable than others. For this mission there will probably be more than one object at any given time, so multi object trackers are a necessity.

3.1.4 Decision

Taking all criteria into consideration, a solution was chosen that significantly differs from usual architectures discussed in this work. The selected configuration consists of an object detector implemented in Tensorflow and Python, YOLO (You Only Look Once) [96] along with a tracker using Deep Metrics and Kalman Filters, Deep Sort [99]. Detections obtained with YOLO are passed to Deep Sort that provides a smooth, more robust tracking with identification of tracked objects. Specifically, a existing open-source solution that integrates a Tensorflow version (darkflow [104]) of YOLO (written in C) with Deep Sort is used, Tracking-with-darkflow [105].

When even reported, it is difficult to compare performance results of different detectors, which obtain their numbers with different hardware and in overall not comparable setups. A team from Google

Research[93] studied this problem and the tradeoff between speed and accuracy for Faster R-CNN, R-FCN, and SSD, obtaining the following results:

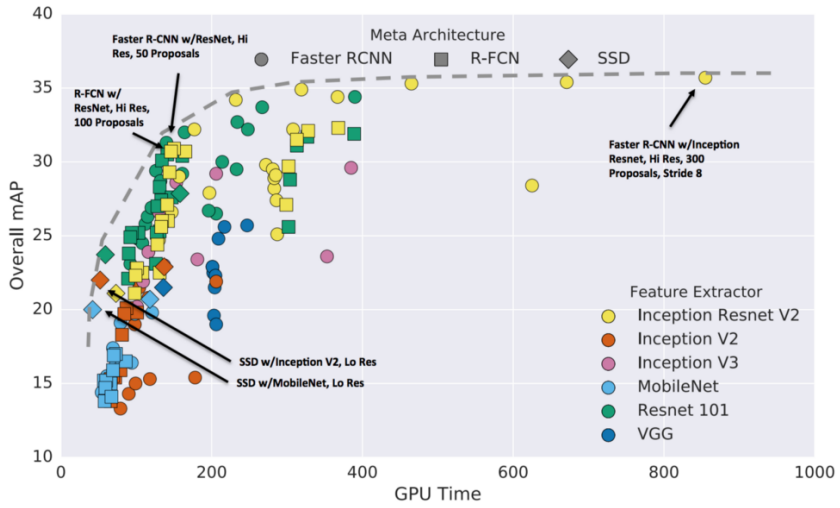


Figure 3.3: Accuracy-speed plot of different configurations of R-FCN, Faster R-CNN and SSD. Taken from [93].

As discussed in Section 2.1, one-stage detectors, though less accurate, are sufficiently fast and accurate for "low-end" hardware and real time operation. Therefore, a detector named YOLO was chosen, for its speed, implementation in Tensorflow and Python, accuracy and support.

3.2 Detection

YOLO is a one-stage detector that divides the image into a $S \times S$ grid. Then, each grid cell predicts bounding boxes and confidence scores for those boxes. The bounding box consists of 5 predictions: x, y, w, h, and confidence. The (x, y) coordinates represent the centre of the box relative to the bounds of the grid cell. Width and height are predicted relative to the whole image. Finally, the confidence prediction represents the IOU (Intersection over Union) between the predicted box and any ground truth box.

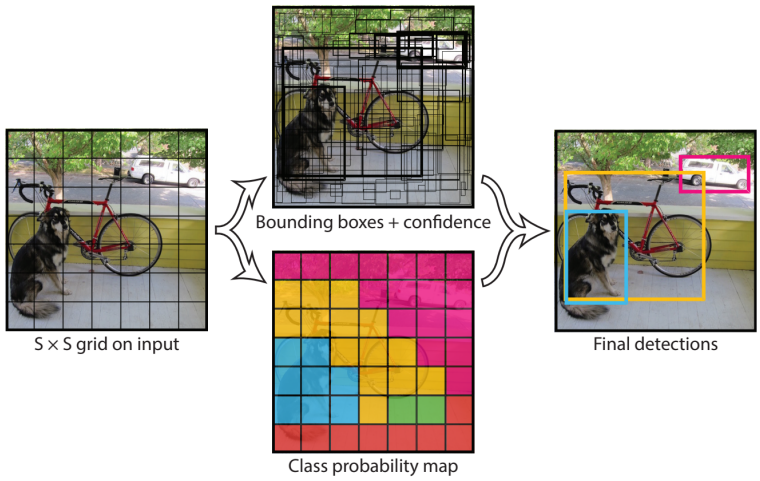


Figure 3.4: YOLO divides the input image in a grid, with each box outputting bounding boxes with confidence scores. Taken from [94].

The network has 24 convolutional layers followed by 2 fully connected layers. Because of its small size this architecture is very fast, and needs to be called only once at test time. Full network is shown in Figure 3.5.

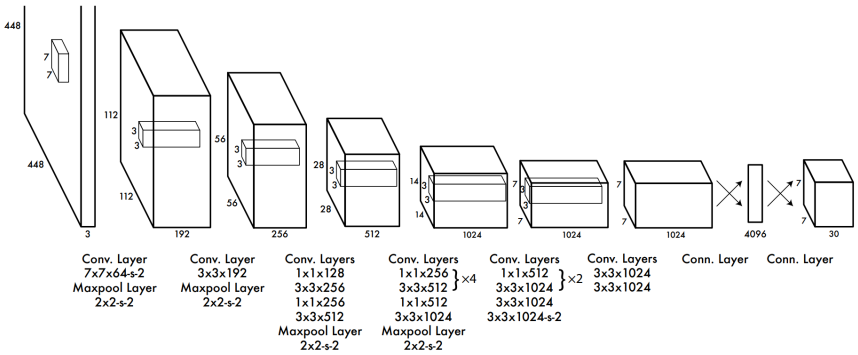


Figure 3.5: Convolutional Neural Network architecture of YOLO detector. Taken from [94].

Note that YOLO is similar in architecture to SSD but is simpler. Versions 2 and 3 of YOLO have made if faster than SSD, though slightly less accurate, depending on input resolution (Figure 3.8).

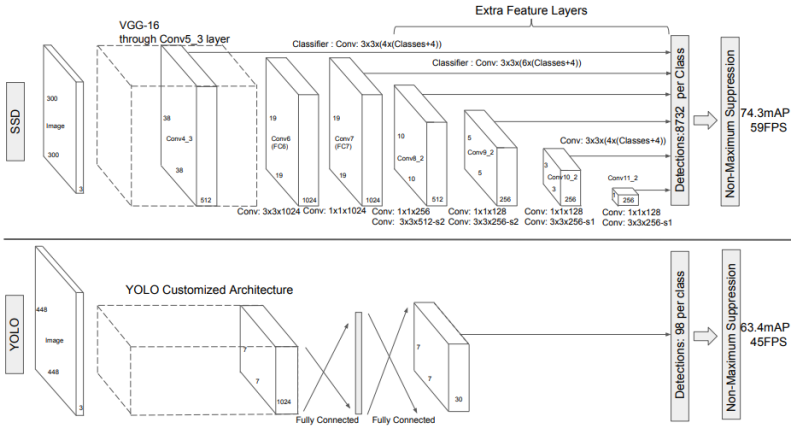
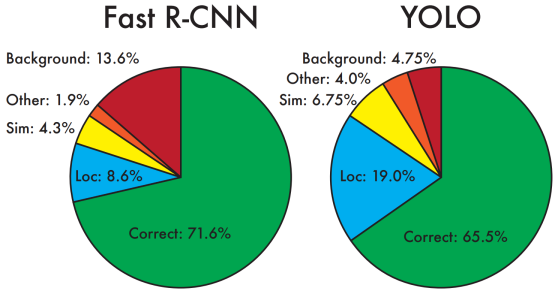


Figure 3.6: Comparison between the architecture of SSD (more complex) and original YOLO. Taken from [67].

YOLO sees the entire input image during training and testing, which allows it to achieve lower background error than other detectors. A breakdown of YOLO error types and comparison with Fast R-CNN is presented in Figure 3.7. This methodology is taken from [97].



Type	Class	IOU
Correct	Correct	IOU > .5
Localization	Correct	.1 < IOU < .5
Similar	Similar	IOU > .1
Other	Wrong	IOU > .1
Background	Any	IOU < .1

Figure 3.7: Error analysis for Fast R-CNN and YOLO. YOLO is slightly less accurate but has lower background error. Including error type definition. Taken from [94].

While YOLO is more imprecise than Fast R-CNN in the spatial location of the bounding box, this is offset by the lower background error and much higher framerate. A performance evaluation is offered in Figure 3.8.

Detection Framework	mAP	FPS
Fast R-CNN	70.0	0.5
Fast R-CNN VGG-16	73.2	7
Fast R-CNN ResNet	76.4	5
YOLO	63.4	45
SSD300	74.3	46
SSD500	76.8	19

Detection Framework		mAP	FPS
YOLOv2	288 x 288	69.0	91
YOLOv2	352 x 352	73.7	81
YOLOv2	416 x 416	76.8	67
YOLOv2	480 x 480	77.8	59
YOLOv2	544 x 544	78.6	40

Figure 3.8: YOLO performance compared to other detectors on PASCAL VOC 2007 with a Geforce GTX Titan X. Taken from [95].

For this work, the version 2 of YOLO will be used, since it offers precision and speed improvements over the original release. Even though version 3 was released shortly before the start of this thesis, compatibility with Darkflow was not yet implemented, so it was not used. Note that YOLOv2 model is scalable, with the same model being able to run at different resolutions. This offers a quick tradeoff between speed and accuracy. Overall it performs on par with state of the art detectors, while at a much higher framerate, even if it incurs in slight imprecisions in the bounding box location. However, one limitation for YOLO is that it only predicts 1 type of class in one grid, hence it struggles with very small objects.

Advantages	Disadvantages
• Very fast	• Localization imprecisions
• Versatile (supports lots of classes)	• Small objects errors
• Training code available	
• Good precision	

3.3 Tracking

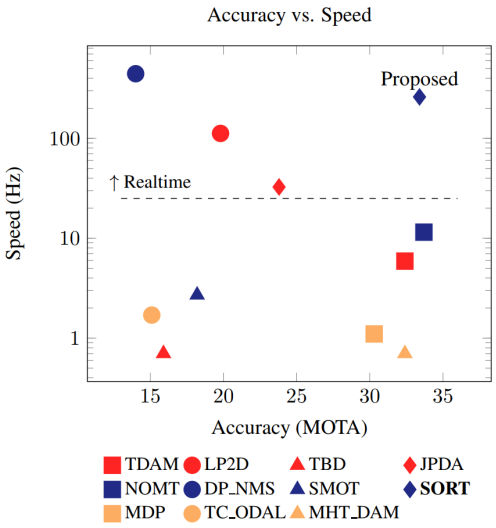


Figure 3.9: Benchmark performance of several trackers including SORT. Taken from [112].

Deep SORT is a multi object tracker evolution of the SORT algorithm. SORT is a relatively simple tracker that uses a combination of techniques such as the Kalman Filter and the Hungarian Algorithm. It is extremely fast (see Figure 3.9) and reliable. At the time of the initial publication (2015), SORT was ranked the best open source multiple object tracker on the MOT benchmark.

SORT receives as input the bounding boxes of the detection of each object, achieving tracking by the tracking-by-detection approach. It focuses in fast, online tracking without incorporating re-identification or occlusion handling. This performance is achieved with a surprisingly simple model.

The state vector of each target is $x = \left[u, v, s, r, \dot{u}, \dot{v}, \dot{s}\right]^T$, where u and v represent the horizontal and vertical pixel location of the centre of the target, while the scale s and r represent the scale (area) and the aspect ratio of the target's bounding box respectively. When a detection is associated to a target, the detected bounding box is used to update the target state where the velocity components are solved with a Kalman filter[100] framework. If no detection is associated to the target, its state is simply predicted without correction using the linear velocity model.

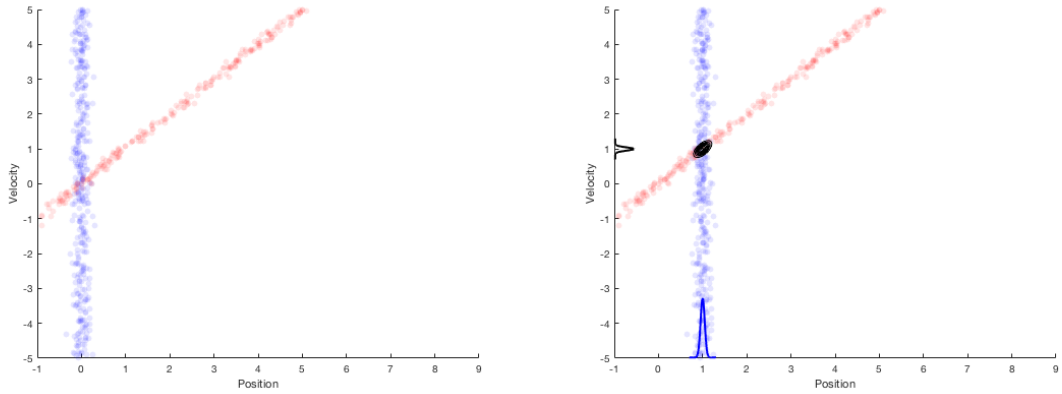


Figure 3.10: An intuitive explanation of Kalman filters: An object is located at position 0 at time 0, with an unmeasured velocity. Blue points denote where it could be located in Velocity-Position axes (left plot). At instant 1 the new position is previous position plus velocity. Therefore, the red points represent where the object could be. If the true velocity of the robot was 1, at instant 1 its position would also be 1. This is represented by the blue points in the image on the right. Again, from basic point mechanics, expected position would be in red region. The overlap between these distributions is small, and Bayes theorem can be applied to estimate the multivariate state distribution probability. Taken from [101].

Kalman filtering is an algorithm that uses a series of measurements observed over time, containing statistical noise and other inaccuracies, and produces estimates of unknown variables,optimal in the special case when the process and measurement follow Gaussian distributions. By estimating a joint probability distribution over the variables for each timeframe, the algorithm tends to be more accurate than those based on a single measurement alone[101]. It is an order of magnitude more computationally expensive than Recursive Least Squares and is based on State-Space models where the entire system needs to be modelled to achieve optimal value, but is more accurate and capable of time series prediction[102].

Detections are assigned to existing targets using an assignment cost matrix computed as the IOU (intersection-over-union), solved optimally with the Hungarian algorithm[103] (an algorithm that solves the problem of assigning tasks to agents with the minimum weight matching in polynomial time). New targets are created when a detection has an overlap less than IOU_{min}. Tracks are terminated if not detected in a number of frames.

This simple algorithm achieved good performance in speed, tracking precision and accuracy, but with a high number of ID switches, due to the fact that occlusions were not dealt with in any explicit way. Deep SORT incorporated several improvements:

- Since distance between predicted and detected bounding boxes can be large when sudden camera motion happens, and barely handles occlusion, a feature comparison metric mas introduced. Using a neural network with two convolutional layers and six residual blocks, the 2.8 million parameters model outputs an appearance descriptor that treated as a cosine distance in the "appearance space". The last hundred descriptors for each track are retained.
- A matching cascade algorithm is introduced to mitigate the probability distribution deterioration during occlusion.

Chapter 4

Training

4.1 Methodology

The general concept of training was explained in Section 1.5, and in this Chapter the process followed in this work will be covered in detail. As the Darfklow version of YOLO is used, it is the training code provided by Darkflow that is employed. The Darfklow port of Darknet is not yet fully completed, so there are differences in how the models are trained, with some Darknet training parameters not used by Darkflow. For example, parameters such as decay-rate and batch subdivision are not implemented (reducing the importance of the validation set, since there are less hyperparameters to tune). This does not make the training process significantly different, but it does mean that using the same dataset and training parameters in Darknet and Darkflow may result in models with slightly different performance, although mostly similar.

UAV123[109] is the dataset chosen to fine tune the YOLOv2 model, with the reasons behind this choice explained in the following Section. Relevant to this Section is, however, the annotations format. This dataset provides them in the format [x1,y1,x2,y2], with (x1,y1) being the coordinates of the top left corner of the bounding box (reference is in the top left corner of the image), and (x2,y2) the coordinates of the bottom right corner. Format required by Darkflow is the PASCAL VOC format, which is a xml tree. Instead of adapting the code to accept other formats, scripts were written to process annotations. Please refer to Appendix A, Section A.1 for more details on formats.

Moreover, while UAV123 annotations only cover one person, there are several sequences where multiple persons appear. This renders these sequences unfit for training. If there are actual persons that are not recognized by the training process it would harm the model's ability to generalize. In order to solve this issue, the tool LabelImg was used to manually annotate the incomplete sequences with the aid of interpolation scripts developed for this purpose. Again, refer to Appendix A for the technical details.

From all the sequences in the dataset only the ones where the focus is in people are selected (26 sequences). 2 sequences, "person21" and "person22" were discarded, since they have too much perspective and size variation with respect to the rest of the sequences (moreover, targets are too small and appear very briefly). A subset of the accepted sequences is separated for training and the remaining for validation.

Unfortunately, training code for Deep SORT's feature extractor was not available until recently, so no training of this model was conducted. However, as it is not a proper object detector, baseline training was sufficient, and no deficiencies were detected that could be associated with this fact. Therefore, only the detector was trained.

4.2 Dataset

Despite the rising number of drones and their applications, there is a lack of datasets and applications of Computer Vision in this field. There is only one dataset not outdated, exclusively recorded from drones, UAV123. A review of the state of the art in aerial datasets, benchmarks and applications is given in the UAV123 paper[109].

The UAV123 dataset is especially fitting to our mission because of its great target size, perspective, illumination and aspect ratio variation, all which are expected on a mobile camera with varying altitude and attitude towards the target. Attribute distribution and comparison with OTB100 dataset is shown in Figure 4.1.

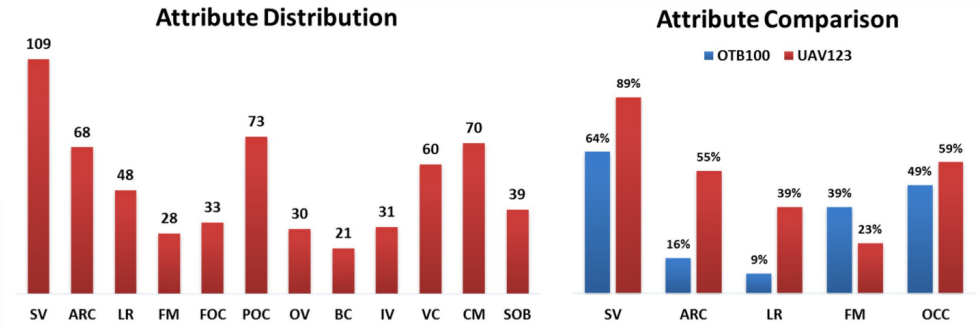


Figure 4.1: Dataset characteristics compared to OTB. Legend is available in Figure 4.2. Taken from [109].

Attr	Description
ARC	Aspect Ratio Change: the fraction of ground truth aspect ratio in the first frame and at least one subsequent frame is outside the range [0.5, 2].
BC	Background Clutter: the background near the target has similar appearance as the target.
CM	Camera Motion: abrupt motion of the camera.
FM	Fast Motion: motion of the ground truth bounding box is larger than 20 pixels between two consecutive frames.
FOC	Full Occlusion: the target is fully occluded.
IV	Illumination Variation: the illumination of the target changes significantly.
LR	Low Resolution: at least one ground truth bounding box has less than 400 pixels.
OV	Out-of-View: some portion of the target leaves the view.
POC	Partial Occlusion: the target is partially occluded.
SOB	Similar Object: there are objects of similar shape or same type near the target.
SV	Scale Variation: the ratio of initial and at least one subsequent bounding box is outside the range [0.5, 2].
VC	Viewpoint Change: viewpoint affects target appearance significantly.

Figure 4.2: Dataset attributes legend. Taken from [109].

This dataset is more varied in aspect ratio, bounding box size and sequence duration than common datasets, as shown in Figure 4.3. Sequence duration hits specially tracker performance (as shown in

Section 2.2) because of tracking drift and, obviously, because the probability of failure increases with time. Aspect ratio challenges the model's idea of "person", since it is associated with perspective, and detectors are sensitive to perspective variations. Bounding box size affects the detection performance of the model as well, but small box sizes will degrade this particular model, YOLO, because of its structure. This was discussed in .Furthermore, sharp variations in perspective and box size poise a challenge for feature extracting/online training trackers, since the object experiences big changes and may be completely unrecognisable for the tracker just seconds apart.

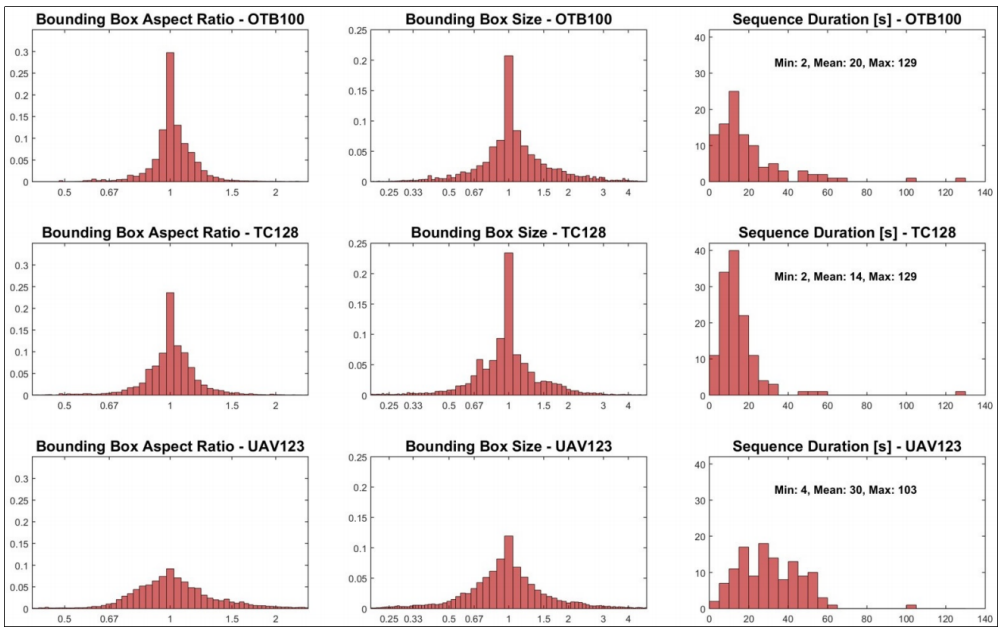


Figure 4.3: Dataset bounding box aspect ratio, box size and sequence duration comparison with OTB100 and TC128. Taken from [109].

Figure 4.4 provides a breakdown of the duration of each sequence. Almost all sequences are longer than 25 seconds, therefore poor tracking metrics are expected. While a number of sequences are low enough to not suffer greatly from this problem, sequences 24, 21, and 9 are disproportionately long with respect to the rest of the dataset, so even worse tracking results are expected.

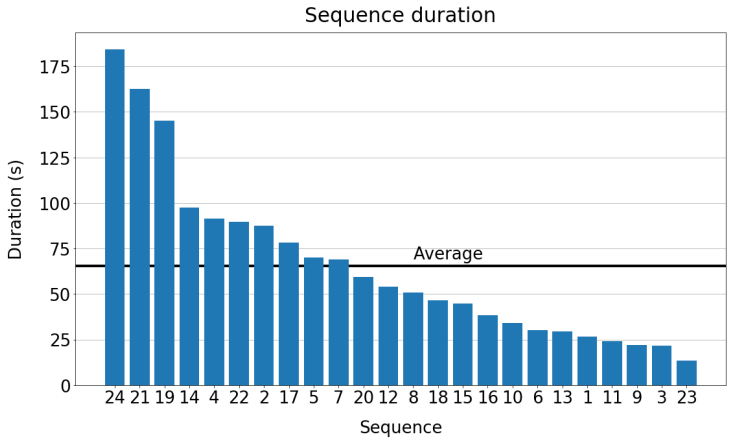


Figure 4.4: Duration in seconds of each sequence in UAV123 dataset in descending order.

One more distinction should be made concerning the number of targets in each sequence. While half of the sequences contain a number of bounding boxes equal or similar to the number of frames (they may be called "single target" at first approximation) the other half shows up to thrice the amount of bounding

boxes. These "multiple target" sequence will probably show worse absolute tracking metrics than the rest and, if the targets are close to each other and far from the camera, will also present lower mAP (because of YOLO).

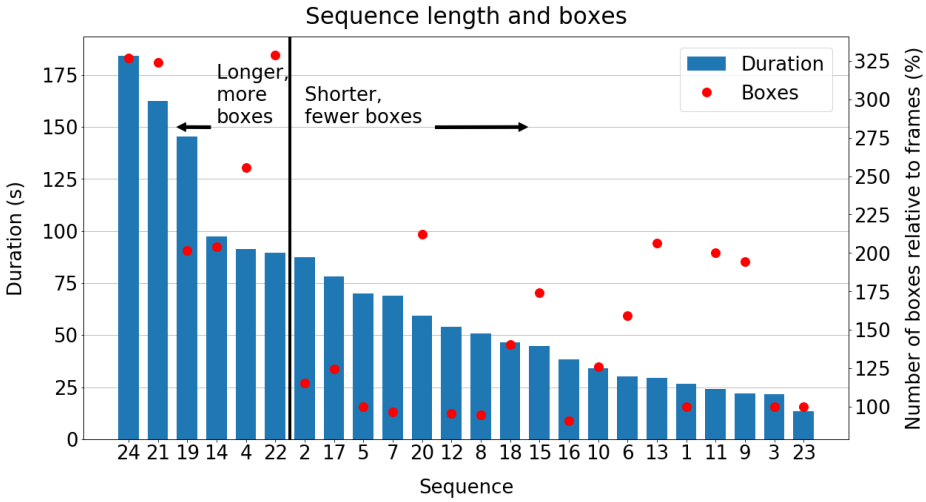


Figure 4.5: Duration in seconds of each sequence in UAV123 dataset in descending order along with the number of bounding boxes relative to the number of frames in each sequence.

Therefore, the group of sequences with the longest duration and higher boxes-to-frames ratio should show especially poor results, while the easier sequences may present acceptable metrics. It is to be noted that, should an individual sequence have peculiarities such as high number of occlusions, variety of colours or small target size, unexpectedly low results may be obtained.

4.3 Process

According to standard practice explained in Section 1.5 the fully annotated dataset consisting of 15472 frames containing 31935 boxes is split into training and validation sets. The latter is formed of sequences 9, 17 and 24, which represent approximately 24% of the dataset.

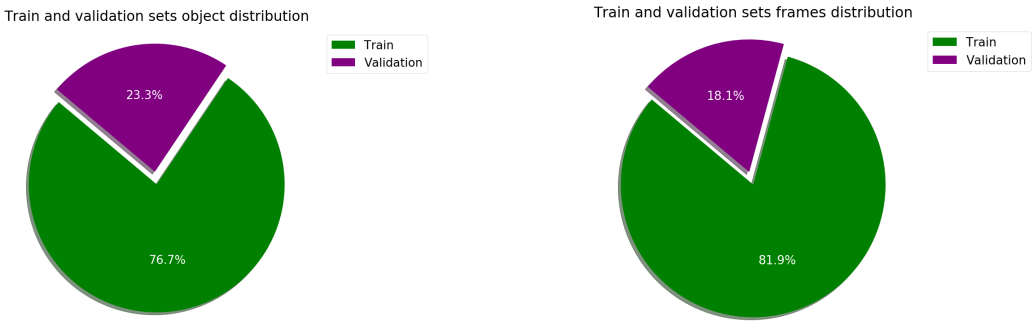


Figure 4.6: Relative sizes of training and validation sets.

The training set is then fed into the network to formally start the training process. Instead of training an empty (random nodes) model, the existing YOLOv2 model is taken and only the last layer is trained, which reduces the magnitude of the task. Therefore, it is only needed to fine tune the model, thus the initial learning rate is set to a relatively low value of 1.10^{-4}. After near three thousand steps, learning rate is reduced to 1.10^{-5}, and near step 9000 is lowered again to 1.10^{-6}.

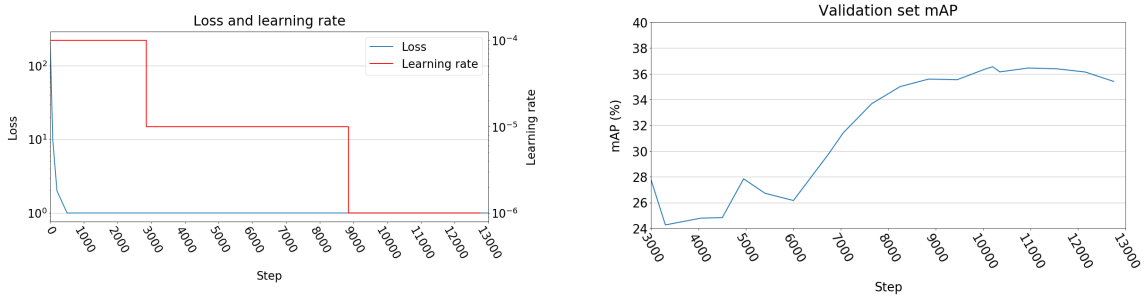


Figure 4.7: Learning rate, value of loss function and validation set mAP during the training process.

Using the validation set mAP chart on the right side of Figure 4.7 to assess model overfitting, it is clear that after step 10000 mAP improvement reached a plateau, and then starts to decline. This is approximately the moment where the training process should be stopped because overfitting is starting to happen. Due to uncertainties in this process, a margin of a thousand steps is taken and the model in step 10950 is selected as the definitive trained model.

4.4 Training Rigour

This training process, while performed according to common practice in the field, leaves room to rigour improvement. First of all, the size could be much greater. Selected sequences represent only a part of the UAV123 dataset, and the already small number of images has to be separated into training and validation sets. There were not enough spare images to form a test set, which is mandatory if the model and its mAP evaluation is to be considered seriously. While datasets such as COCO or ImageNet have images in the range of hundreds of thousands or even millions, this training was conducted with approximately 15 thousand images.

A more subtle concern is that of sequence similarity. Some sequences appear to have been recorded together, with the same targets appearing on screen (wearing the same clothes, in the same scenarios). Greater average sequence duration means that there is a comparatively low number of sequences, which aggravates the problem. This makes overfitting specially worrying, thus justifying the expense of images in the validation set even with in this scarce setting. However, sequence similarity also affects the validation set, as targets in this set may share aspect and scenario with those in the training set. While not an exact match, the accuracy of overfitting prediction of the validation set is reduced.

Nevertheless, this process is rigorous enough for the purpose of this work, which is not detection but tracking. Dataset size is small but acceptable (PASCAL VOC is of similar size, even though is small by today's standards) and, while the model may be overfitting despite efforts to prevent it, it is not of paramount importance, as this is only a way of providing the tracker with acceptable detections. They could have been provided with another model, and achieved tracking results would be exactly as valid.

Chapter 5

Mission

5.1 Mission Architecture

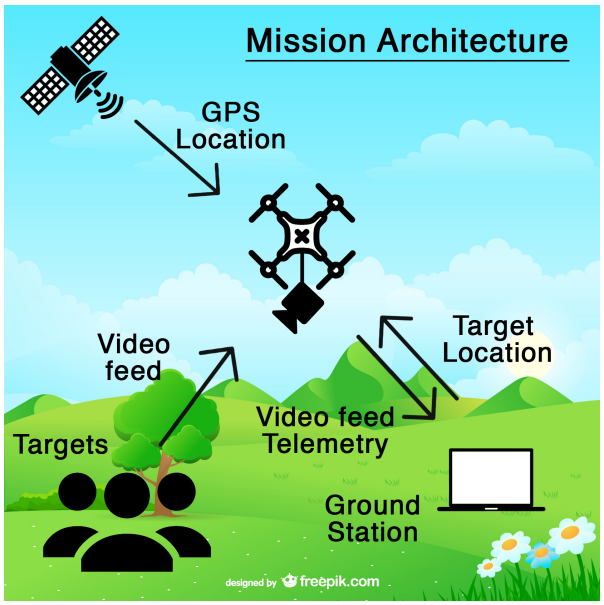


Figure 5.1: Infographic illustrating proposed mission architecture.

Mission objective is to follow a desired target with a UAV. In order to achieve this, an Artificial Intelligence Computer Vision algorithm is used, more specifically, a Convolutional Neural Network detector along with a Machine Learning tracker. The drone is managed through a ground station, which also processes the video feed gathered from the drone. Target location is obtained as output of the tracking software, which is sent back to the UAV, which would proceed to follow the target with its auto-pilot.

5.2 Tests

As the source of the video feed is actually irrelevant to the tracker, all operations in this work were performed with a rendered video or even raw images as input. However, some steps into simulating the real mission environment were made. The most important of them was preliminary work with a Raspberry Pi 3, which was used as video source.

In order to control the drone, QGroundControl[132] or another similar program would be used. These programs usually establish automatic connection with the drone with minimal setup, and are capable of

managing take off and landing, flight routes and waypoints with GPS location and receiving video feeds. Connection is established through either UDP, RTP or RTSP, which are network protocols part of the Internet Protocol Suite, also known as TCP/IP (originally funded by DARPA[133]). To simplify it, this can be thought of as QGroundControl and the drone establishing a private WiFi connection between them.

There are two ways of passing the video feed to the tracker: Getting the video feed from QGroundControl, where it is inside a hardcoded h.264 pipeline (which requires advanced knowledge of Qt to manipulate), or manually establishing a separate streaming directly to the tracker. In order to simulate the latter, a setup similar to that of Figure 5.2 was used.

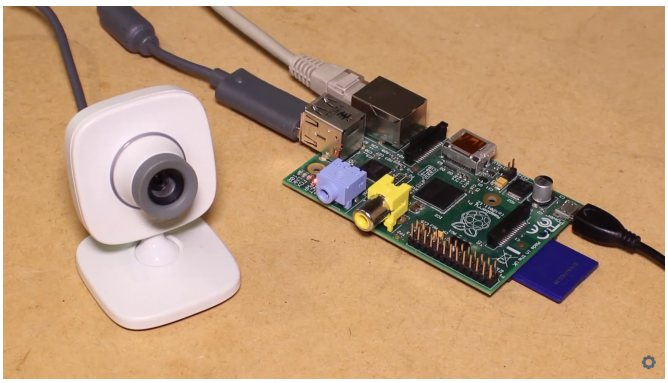


Figure 5.2: Example setup of a Raspberry Pi with a USB camera and Ethernet connection. Taken from [136]

With the Raspberry representing the on-board drone computer, a USB camera provided the video source. Video was streamed through a HTTPS stream (which is just another protocol such as RTP or RTSP - UDP belongs to the transport layer like TCP) using mjpg-streamer[135]. Both the Raspberry and the computer hosting the tracker were in the same local network.

The test was a success and the tracker correctly processed the video feed from the USB camera connected to the Raspberry. The only remaining link of communication to be explored is how to send to the drone relevant telemetry such as target location either in screen coordinates or geo-locating its position knowing the precise location of the drone and estimating its position relevant to the camera.

Chapter 6

Results

The primary result of this work is to improve the performance of the existing Darkflow + DeepSORT tracker in video taken from a drone. Initial tests with the UAV123 dataset evidence unsatisfactory performance, with the main cause being scarce and unreliable detections. In order to solve this issue, Darkflow was trained on the UAV123 dataset. Gained performance will be evaluated in this chapter.

6.1 Original Detection Performance

First of all, the original model detection performance has to be assessed to compare it to the results obtained in the trained model. The relevant metric for detectors is Mean Average Precision and is obtained with an open source project [131]. This metric and the justification behind it were discussed in Section 2.3. mAP of each individual sequence is shown in Figure 6.1.

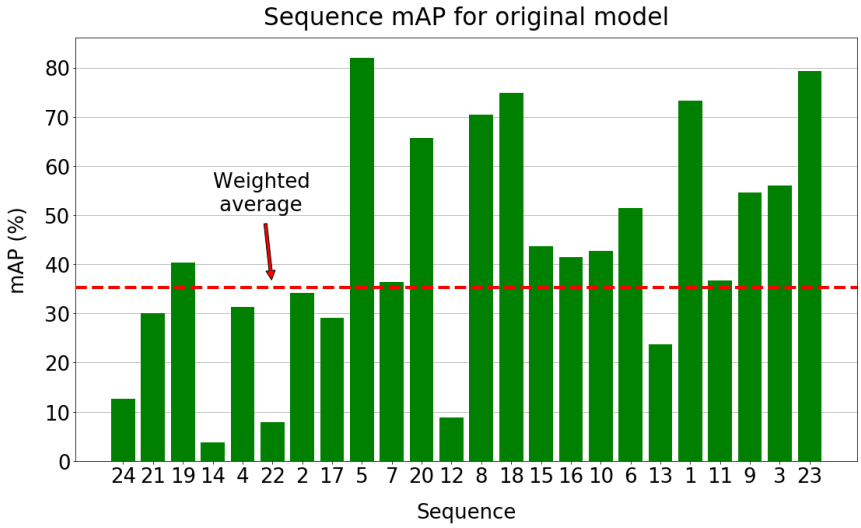


Figure 6.1: mAP of each sequence used of the UAV123 dataset obtained with the original, untrained YOLO model. Average weighted with sequence duration is included.

Note that the weighted average mAP of a set of sequences is not equal to the mAP obtained if all images on the sequences would be treated as an unique sequence. This is due to the correction made when computing the integral of the Precision-Recall curve, which considers Precision to decrease monotonically. Although the latter method is the correct one, the former will be used, since the difference is very small (2-3%), computing all sequences at once is expensive and this result is secondary.

Therefore, the weighted average mAP is 35.34%, which is low. Though not strictly comparable because it was achieved with different datasets, YOLOv2 mAP for COCO data set is 48.1%, with YOLOv3

obtaining up to 57.9%. Other detectors are in the 45-55% range, and for easier datasets (such as PASCAL VOC 2012) YOLOv2 and other detectors achieved 70% mAP. This highlights the difficulty in using models trained in "standard view" in "aerial view" setings.

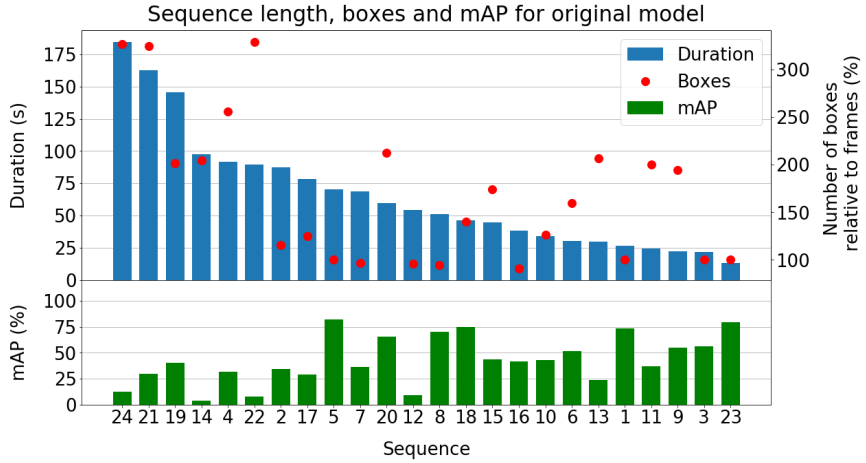


Figure 6.2: mAP of each sequence along with sequence duration and boxes plot for the original, untrained model.

In Chapter 4 poorer tracking metrics were predicted for a group of sequences that were the longest and also contained a high relative number of bounding boxes. However, number of targets and number of images are irrelevant to the detection task. Figure 6.2 shows sequence length and mAP plots together, and the unwanted effect is noticeable. Two reasons explain this problem:

- Some sequences were recorded as "person" sequence, where there is only a single main target, and others were recorded as "group" sequences, where there are at least three main targets. These "group" targets usually walk close to each other and, as they have small size on screen, YOLO struggles to detect them individually (as mentioned in Section 3.2).

- At first, main problem seems to be sequence duration. Coincidentally, the longest sequences also have the largest number of bounding boxes per frame. Poor performance in short sequences such as 11 and 13 would prove a stronger association with the number of boxes. However, these are weak correlations, because the main effect in mAP is actually distance between camera and target. Longer sequences have more varied viewpoints and perspective changes, whereas shorter sequences often consist of simpler shots, most of the time closer to the target.

6.2 Original Tracking Performance

After evaluating the performance of the original detector model, the 24 sequences were fed into the tracker (with the original model) and its output analysed. Official MOT Challenge benchmark tools[120] are open source, and their python implementation[121] was used. Then, the metrics for each sequence were averaged proportionally to their duration, and results are shown in Figure 6.3:

IDF1	Recall	Precision	GT	MT	PT
31.47%	58.78%	97.66%	5.26	0.52	3.11

ML	FP	FN	IDS	FM	MOTA
1.63	21.32	1194.33	45.05	115.98	55.84%

Figure 6.3: Sequence duration weighted average tracking metrics for original model. GT denotes Ground Truth target trajectories. Obtained with a threshold of IOU = 0.5. Average sequence duration is 654 frames, or 65 seconds.

These metrics should be thought of as describing an imaginary sequence with average duration and ground truth targets. First of all, there are several detection metrics in the table: Recall, Precision, FP and FN. Precision (97.66%) is excellent, and false positives are low enough. However, false negatives are inadequate (almost two false negatives per frame) and Recall (58.78%) is disappointing. This shows that the tracker has had a hard job distinguishing if detections belonged to any real target.

For the proper tracking metrics, this average sequence has 5.26 targets, of which 0.52 (9.8%) are mostly tracked, 1.63 (30.9%) mostly lost, and the remaining partially tracked. These are perhaps the most easy to understand, straightforward tracking metrics, and portray poor results. The more abstract MOTA and IDF1 are relatively low, but MOTA would be just in the acceptable range. However, fragmentations and ID switches are deficient. A fragmentation occurs every 6 frames, and on average a target gets its ID switched 9 times.

In order to compare these results more easily with the metrics that will be obtained with the trained model, a number of metrics are selected to clearly describe performance. These are IDF1, Recall, Precision, MT, FN, IDS, FM and MOTA. To make plotting these metrics together more comfortable, dimensional metrics (MT, FN, IDS and FM) will be nondimensionalized. Proposed nondimensionalization are detailed in Figure 6.4.

Metric	Nondimensionalization
MT	$\frac{MT}{GT} = \frac{Mostly\ Tracked}{Ground\ Truth}$
FN	$\frac{FN}{\sum_{i=1}^{24} Length_i \times GT} = \frac{False\ Negatives}{Average\ Sequence\ Duration\ \times Ground\ Truth}$
IDS	$\frac{IDS}{GT^3} = \frac{ID\ Switches}{(Ground\ Truth)^3}$
FM	$\frac{FM}{GT^3} = \frac{Fragmentations}{(Ground\ Truth)^3}$

Figure 6.4: Sequence duration weighted average tracking metrics for original model. MOTP not included because correct computation is not yet implemented in employed library.

Since MT is of the same order of magnitude as GT, this nondimensionalization is straightforward. False negatives in a frame are of the same order of magnitude as GT, so number of frames times GT is the logic nondimensionalization. IDS and FM do not have a clear correlation with any other metric, but in the range of numbers of this dataset a nondimensionalization with GT cubed is near the desired order of magnitude of 1, therefore it is used. Note that, especially for these two metrics (where lower is better),

the focus is on obtaining numbers that are easy to read rather than theoretical proof (that does not exist) or behaviour in extreme values. Resulting metrics are pictured in Figure 6.5:

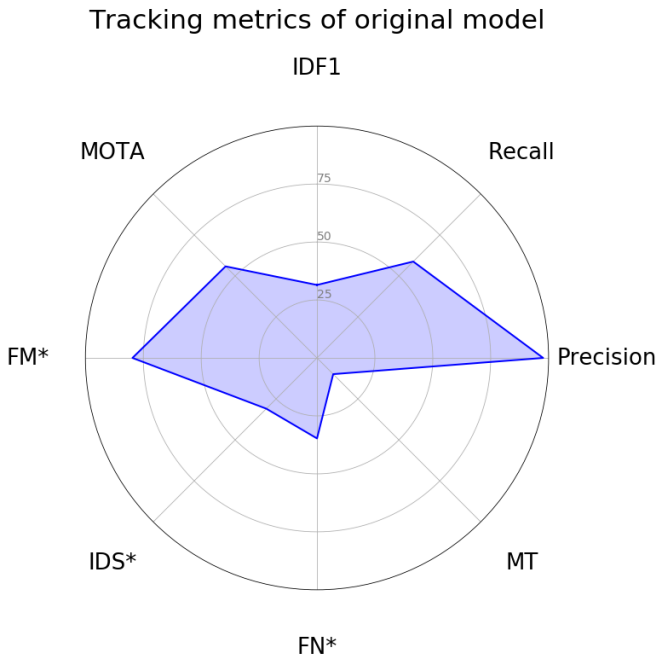


Figure 6.5: A selection of tracking metrics of the original model. Note that metrics with an asterisk indicate better performance the lower they are.

6.3 Trained Detection Performance

Once the training process is completed detection performance is evaluated again, now with the distinction between train and validation sets.

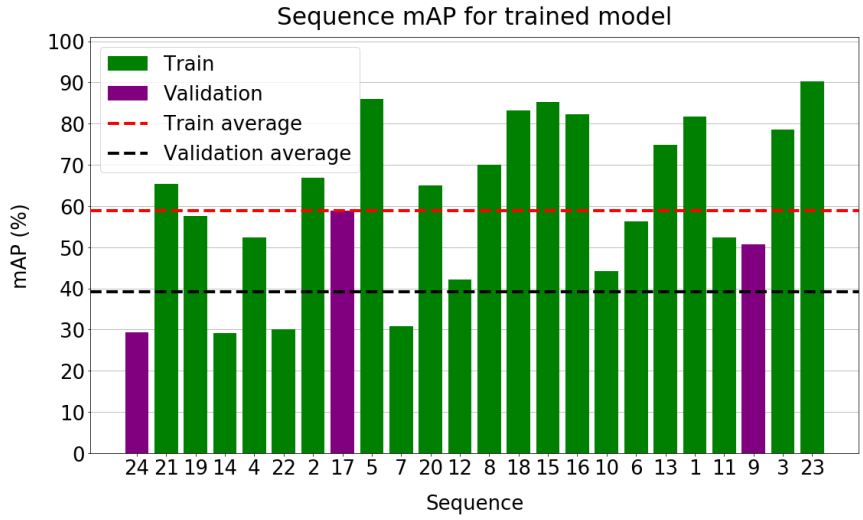


Figure 6.6: mAP of each sequence used of the UAV123 dataset obtained with the trained YOLO model. Average weighted with sequence duration is included for train and validation sets.

Major precision gains are achieved, as it is easily noticeable. mAP for all the sequences is 55.22%, with 58.78% for the training set and 39.14% for the validation set. These numbers have little value on their own, for the validation set is small and no insight into how much the model is overfitting is

obtained. Again, this is not important, because the real effect to be measured in this work is the tracking performance, which required an acceptable detector. If true performance of this model is to be assessed, it should be evaluated in an entirely different dataset.

Back to Figure 6.6, no sequence has mAP lower than 30%, and values up to 80% are reached in the easier sequences. mAP in the validation set is lower as expected. Improvement for each sequence is shown in Figure 6.7.

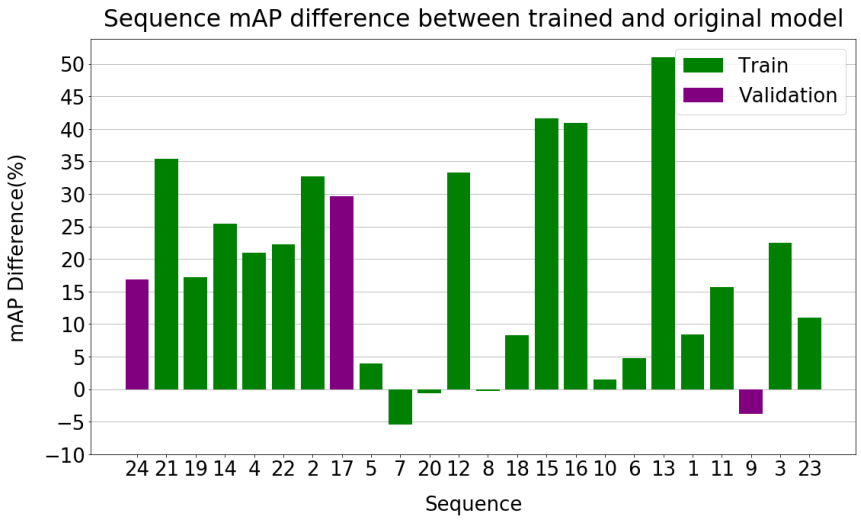


Figure 6.7: mAP gain of each sequence compared to the untrained model.

Sequences that gained the most mAP were those that started with lower mAP. Though it may seem surprising that some sequences barely gained or even lost mAP, those sequences had good mAP in the original model, thus model limit is reached. If extremely high mAP were achieved in a training process it would mean either that the dataset is trivially easy (which is not the case) or that the model is overfitting, which is undesirable. In this case, 2 validation set sequences gained mAP in the same order of magnitude than those in the training set, while sequence 9 lost about 4% mAP, but it already had 50.74% mAP in the original model, so it is not alarming.

6.4 Trained Tracking Performance

With the trained model loaded on the tracker, tracking metrics are once again computed:

IDF1	Recall	Precision	GT	MT	PT
41.12%	82.86%	98.34%	5.26	2.62	2.39

ML	FP	FN	IDS	FM	MOTA
0.24	31.67	521.83	52.78	75.98	79.48%

Figure 6.8: Sequence duration weighted average tracking metrics for trained model. GT denotes Ground Truth target trajectories. Obtained with a threshold of IOU = 0.5. Average sequence duration is 654 frames, or 65 seconds.

Detection metrics have greatly improved, as expected, with Recall at 82.86% and Precision maintaining its excellent value. False positives are slightly higher, but false negatives have sharply dropped

to half its previous value. For tracking metrics, MT has quadrupled, and now one out of two targets are mostly tracked throughout the sequence, while Mostly Lost has fallen to barely 10% of the targets. A comparative between the original and trained models is given in Figure 6.9:

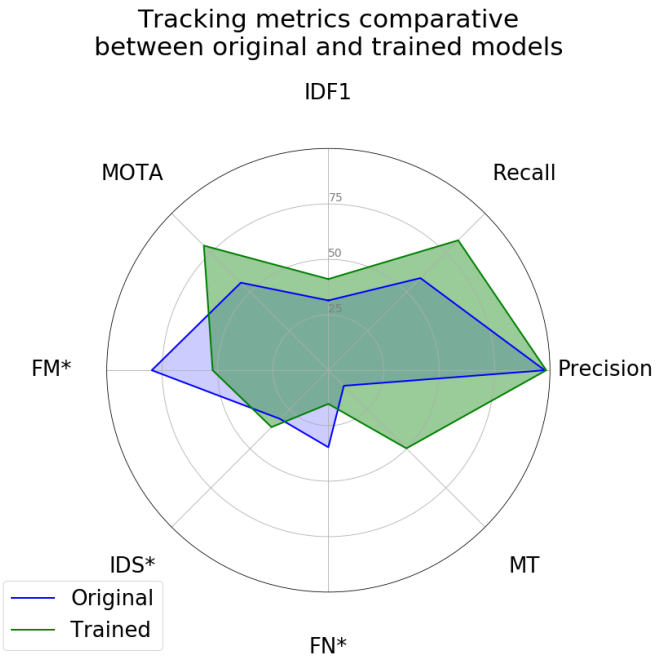


Figure 6.9: A comparative of a selection of tracking metrics between original and trained models. Note that metrics with an asterisk indicate better performance the lower they are.

At first glance, biggest improvements were obtained in MT, Recall, MOTA and FM, with IDF1 and FN only slightly improving, Precision remaining constant, and IDS deteriorating. However, such regress is deceitful. Even though there are much more consistent trajectories and better detections, IDS rose only slightly, which can be considered as a success.

Chapter 7

Conclusions

Autonomous tracking from an aerial platform using Deep Learning is advanced in this work with the design of an architecture and fine-tuning of the detector model in aerial settings. However, fully autonomous operation is not yet possible with the current state of the art and available open source techniques.

7.1 Model Readiness For Mission

For this mission, the following values would be desired for these metrics:

- Precision: 100%. All targets have to be almost constantly detected by the model if the tracker is to have significant chances of successfully performing its job. Kalman Filters only improve uncertainties, they cannot substitute a detector and will struggle to work with a deficient one.
- MT: 100%. All targets should be tracked throughout the mission at almost every instant.
- FM: 0. This requisite may be softer than the other, because short fragmentations can probably be assimilated by the system. The target will probably stay in a close enough range for re-targetting to happen if the trajectory fragmentation is short. Some auto-pilot strategies could be implemented to mitigate this issue, such as maintaining extrapolated target trajectory for a few seconds. Nevertheless, FM must be kept low.
- IDS: 0. This is the most demanding requisite, but it is mandatory for autonomous operation. Every time a ID Switch happens, ground station operator has to intervene and manually select the new ID to continue the mission, which is undesirable.

Achieved results of this work are insufficient. For the trained model, on average an ID Switch happens every ten seconds. Real world operation of this system would demand IDS frequencies below one every several minutes, therefore a ten-fold improvement on this metric is required. This is the reason why, while MOTA may be a widespread metric useful for comparing different trackers and portraying improvements, IDF1 is a much better metric for illustrating the real state of this technology. When IDF1 scores greater than 90% are reached, systems such as this one will probably be ready for real world operation.

7.2 Future Work

To address the shortcomings of this architecture, the following investigation lines are recommended.

7.2.1 A Different Tracker

Many alternatives to the configuration used in this work exist. However, a number of problems arise when considering what tracker to use for this task. As has been discussed before, open-source and hardware needed to perform in real time are the main obstacles. Moreover, it is difficult to assess beforehand a tracker's performance and precision without actually testing it. Evaluation metrics are not fully standardized, and performance measurements are often performed in different hardware setups. Speed comparison is made even less reliable by the fact that different trackers may be optimized for different setups, with some requiring large amounts of memory or multi-core systems. For example, in Figure 2.42, speed data was not obtained by MOT Benchmark organizers and was instead supplied by the trackers' creators, each with a different hardware configuration and optimization skills.

Nevertheless, a number of promising open source trackers[1] exist. Using Caffe framework would allow the use of detectors such as the slightly more accurate SSD (Faster R-CNN is most probably not fast enough for online tracking) and trackers such as GOTURN. Trackers written for MATLAB[2] may perform similarly but integration with the rest of the software would be much more difficult. A number of trackers to consider can be found in collection sites[3][4][5][6].

7.2.2 A New Dataset

UAV123 dataset aligns well with the objectives of this work, but unfortunately contains too few images for a proper training. The most recognized drone dataset, Stanford Drone Dataset[7], aims for a different viewpoint than that of this work, so it is not useful. Therefore, a new dataset could be created from scratch using a drone-mounted camera. Annotation would be the painful part of this proposal and, though efforts are being made[134], the problem remains at large unsolved.

Crowd-labelling could be used, but the proposed method for annotating this dataset would be using the model trained in this work as preliminary annotations, which would then be perfected by human labellers.

7.2.3 A New Tracker

The used tracker, SORT, is incredibly simple in concept. Using the tracking-by-detection approach, a similar tracker could be easily developed if the Kalman Filter and Hungarian algorithm are mastered. Though Deep SORT adds feature extraction, it is not trajectory regression or identity assignment that should be the focus (these two problems are largely resolved) but feature descriptors for re-identification. Either building a new tracker or expanding on Deep SORT, there is enough performance margin available to invest in improved feature extraction.

[1] https://github.com/bochinski/iou-tracker/

[2] https://bitbucket.org/amilan/segtracking

[3] https://github.com/kjw0612/awesome-deep-vision

[4] https://github.com/abhineet123/Deep-Learning-for-Tracking-and-Detection

[5] https://github.com/handong1587/handong1587.github.io/blob/master/_posts/deep_learning/2015-10-09-tracking.md

[6] https://github.com/foolwood/benchmark_results

[7] http://cvgl.stanford.edu/projects/uav_data/

Bibliography

[1] S. Russel, P. Norvig. Artificial Intelligence: A Modern Approach. 2009. Prentice Hall Press. http://aima.cs.berkeley.edu/

[2] M. Minsky. It's 2001. Where Is HAL?. Dr. Dobb's Technetcast. 2001.

[3] R. Kurzweil. The Singularity is Near. Viking Press. 2005.

[4] J. Hawkins, S. Blakeslee. On Intelligence. NY: Owl Books. 2004.

[5] C. McClelland. The Difference Between Artificial Intelligence, Machine Learning, and Deep Learning. https://medium.com/iotforall/the-difference-between-artificial-intelligence-machine-learning-and-deep-learning-3aa67bff5991

[6] The Turing Test. Department of Psychology of University of Toronto. http://www.psych.utoronto.ca/users/reingold/courses/ai/turing.html

[7] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers, *in IBM Journal of Research and Development*. 1959.

[8] D. Soni. Supervised vs. Unsupervised Learning. https://towardsdatascience.com/supervised-vs-unsupervised-learning-14f68e32ea8d

[9] Introduction to Deep Neural Networks (Deep Learning). https://deeplearning4j.org/neuralnet-overview

[10] D. Sussillo. Random Walks: Training Very Deep Nonlinear Feed-Forward Networks with Smart Initialization. 2014. arXiv:1412.6558.

[11] Using neural nets to recognize handwritten digits. http://neuralnetworksanddeeplearning.com/chap1.html

[12] N. Akhtar, A. Mian. Threat of Adversarial Attacks on Deep Learning in Computer Vision: A Survey. 2018. arXiv:1801.00553.

[13] A. Tchircoff. The mostly complete chart of Neural Networks, explained. https://towardsdatascience.com/the-mostly-complete-chart-of-neural-networks-explained-3fb6f2367464

[14] A. Mai Nguyen, J. Yosinski, J. Clune. Multifaceted Feature Visualization: Uncovering the Different Types of Features Learned By Each Neuron in Deep Neural Networks. 2016. arXiv:1602.03616.

[15] M. T. Ribeiro, S. Singh, C. Guestrin. Why Should I Trust You?: Explaining the Predictions of Any Classifier. 2016. arXiv:1602.04938.

[16] A. Kurakin Adversarial Attacks and Defences Competition. 2018. arXiv:1804.00097.

[17] J. Su, D. V. Vargas, K. Sakurai. One pixel attack for fooling deep neural networks. 2017. arXiv:1710.08864.

[18] G. F. Elsayed, S. Shankar, B. Cheung, N. Papernot, A. Kurakin, I. J. Goodfellow, J. Sohl-Dickstein. Adversarial Examples that Fool both Human and Computer Vision. 2018. arXiv:1802.08195.

[19] N. Donges. Recurrent Neural Networks and LSTM. https://towardsdatascience.com/recurrent-neural-networks-and-lstm-4b601dd822a5

[20] A. Graves, A. Mohamed, G. Hinton. Speech recognition with deep recurrent neural networks, *in Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference pages 6645-6649*. 2013.

[21] E. W. Weisstein. Convolution. From MathWOrld - A Wolfram Web Resource. http://mathworld.wolfram.com/Convolution.html

[22] D. Cornelisse. An intuitive guide to Convolutional Neural Networks. https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050

[23] U. Karn. An Intuitive Explanation of Convolutional Neural Networks. https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/

[24] A. Dertat. Applied Deep Learning - Part 4: Convolutional Neural Networks. https://towardsdatascience.com/applied-deep-learning-part-4-convolutional-neural-networks-584bc134c1e2

[25] A Beginner's Guide to Deep Convolutional Neural Networks (CNNs). https://deeplearning4j.org/convolutionalnetwork

[26] J. Mugan. What Deep Learning Really Means. https://www.linkedin.com/pulse/20141114065942-42285562-what-deep-learning-really-means/

[27] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, Y. Bengio. Generative Adversarial Networks. 2014. arXiv:1406.2661.

[28] T. Silva. An intuitive introduction to Generative Adversarial Networks (GANs). https://medium.freecodecamp.org/an-intuitive-introduction-to-generative-adversarial-networks-gans-7a2264a81394

[29] A. Radford, L. Metz, S. Chintala. Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks. 2015. arXiv:1511.06434.

[30] T. Karras, T. Aila, S. Laine, J. Lehtinen. Progressive Growing of GANs for Improved Quality, Stability, and Variation. 2017. arXiv:1710.10196.

[31] M. S. M. Sajjadi, B. Schölkopf, M. Hirsch. EnhanceNet: Single Image Super-Resolution through Automated Texture Synthesis. 2016. arXiv:1612.07919.

[32] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, M. Nießner. Face2Face: Real-Time Face Capture and Reenactment of RGB Videos, *in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2387-2395*. 2016. Las Vegas, NV.

[33] S. Suwajanakorn, S. M. Seitz, I. Kemelmacher-Shlizerman. Synthesizing Obama: Learning Lip Sync from Audio. 2017.

[34] A. Rössler, D. Cozzolino, L. Verdoliva, C. Riess, J. Thies, M. Nießner. FaceForensics: A Large-scale Video Dataset for Forgery Detection in Human Faces. 2018. arXiv:1803.09179.

[35] K. He, X. Zhang, S. Ren, J. Sun. Deep Residual Learning for Image Recognition. 2015. arXiv:1512.03385.

[36] S. Zagoruyko, N. Komodakis. Wide Residual Networks. 2016. arXiv:1605.07146.

[37] Z. Luo. Different Tasks in Computer Vision. https://luozm.github.io/cv-tasks

[38] A. Kirillov, K. He, R. B. Girshick, C. Rother, Piotr Dollár. Panoptic Segmentation. 2018. arXiv:1801.00868.

[39] F. I. Alam et al. Conditional Random Field and Deep Feature Learning for Hyperspectral Image Segmentation. 2017. arXiv:1711.04483.

[40] K. Janocha, W. Marian Czarnecki. On Loss Functions for Deep Neural Networks in Classification. 2017. arXiv:1702.05659.

[41] C. Bourez. About loss functions, regularization and joint losses. http://christopher5106.github.io/deep/learning/2016/09/16/about-loss-functions-multinomial-logistic-logarithm-cross-entropy-square-errors-euclidian-absolute-frobenius-hinge.html

[42] R. Shukla. L1 vs L2 Loss function. http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/

[43] Differences between L1 and L2 as Loss Function and Regularization. http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/

[44] Loss Functions. http://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

[45] S. Ruder. An overview of gradient descent optimization algorithms. 2016. arXiv:1609.04747.

[46] S. Sharma. Epoch vs Batch Size vs Iterations. https://towardsdatascience.com/epoch-vs-iterations-vs-batch-size-4dfb9c7ce9c9

[47] A. Bhande. What is underfitting and overfitting in machine learning and how to deal with it. https://medium.com/greyatom/what-is-underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6803a989c76

[48] Hyperparameters. http://www.brnt.eu/phd/node14.html

[49] CUDA. https://developer.nvidia.com/cuda-zone

[50] M. Abadi et al. TensorFlow: A system for large-scale machine learning, *in 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16), pages 265-283*. 2016.

[51] A. Sachan. Zero to Hero: Guide to Object Detection using Deep Learning: Faster R-CNN,YOLO,SSD. http://cv-tricks.com/object-detection/faster-r-cnn-yolo-ssd/

[52] N. Dalal and B. Triggs. Histograms of Oriented Gradients for Human Detection, *in International Conference on Computer Vision and Pattern Recognition (CVPR '05), pages 886-893*. 2005.

[53] A. Krizhevsky, I. Sutskever, G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks, *in Proceedings of the 25th International Conference on Neural Information Processing Systems, Vol. 1, pages 1097-1105*. 2012.

[54] C. Szegedy et al. Going Deeper with Convolutions. 2014. arXiv:1409.4842.

[55] S. Das. CNN Architectures: LeNet, AlexNet, VGG, GoogLeNet, ResNet and more .... https://medium.com/@siddharthdas_32104/cnns-architectures-lenet-alexnet-vgg-googlenet-resnet-and-more-666091488df5

[56] G. Seif. Deep Learning for Image Recognition: why it's challenging, where we've been, and what's next. https://towardsdatascience.com/deep-learning-for-image-classification-why-its-challenging-where-we-ve-been-and-what-s-next-93b56948fcef

[57] J. Xu. An Intuitive Guide to Deep Network Architectures. https://towardsdatascience.com/an-intuitive-guide-to-deep-network-architectures-65fdc477db41

[58] F. Chollet Xception: Deep Learning with Depthwise Separable Convolutions. 2016. arXiv:1610.02357.

[59] R. B. Girshick, J. Donahue, T. Darrell, J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. 2013. arXiv:1311.2524.

[60] D. Parthasarathy. A Brief History of CNNs in Image Segmentation: From R-CNN to Mask R-CNN. https://blog.athelas.com/a-brief-history-of-cnns-in-image-segmentation-from-r-cnn-to-mask-r-cnn-34ea83205de4

[61] R. B. Girshick. Fast R-CNN. 2015. arXiv:1504.08083.

[62] S. Ren, K. He, R. B. Girshick, J. Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. 2015. arXiv:1506.01497.

[63] J. Xu. Deep Learning for Object Detection: A Comprehensive Review. https://towardsdatascience.com/deep-learning-for-object-detection-a-comprehensive-review-73930816d8d9

[64] J. Dai, Y. Li, K. He, J. Sun. R-FCN: Object Detection via Region-based Fully Convolutional Networks. 2016. arXiv:1605.06409.

[65] K. He, G. Gkioxari, P. Dollár, R. B. Girshick. Mask R-CNN 2017. arXiv:1703.06870.

[66] R. B. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, K. He. Detectron. 2018. https://github.com/facebookresearch/detectron

[67] W. Liu et al. SSD: Single Shot MultiBox Detector. 2016.

[68] T. Lin, P. Goyal, R. B. Girshick, K. He, P. Dollár. Focal Loss for Dense Object Detection. 2017. arXiv:1708.02002.

[69] OpenCV API Documentation. Hough Line Transform. Structural Analysis and Shape Descriptors. https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_houghlines/py_houghlines.html

[70] OpenCV API Documentation. Structural Analysis and Shape Descriptors. https://docs.opencv.org/3.0-beta/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

[71] Valentine Vega, Désiré Sidibé, Yohan Fougerolle. Road Signs Detection and Reconstruction using Gielis Curves, *in International Conference on Computer Vision Theory and Applications, pages 393-396*. 2012. Rome.

[72] M. B. Hisham, Shahrul Nizam Yaakob, Raof R. A. A, A.B A. Nazren, N.M.Wafi. Template Matching using Sum of Squared Difference and Normalized Cross Correlation, *in IEEE Student Conference on Research and Development (SCOReD)*. 2015.

[73] N. Sadat Hashemi, R. Babaie Aghdam, A. Sadat Bayat Ghiasi, P. Fatemi. Template Matching Advances and Applications in Image Analysis. 2016. arXiv:1610.07231.

[74] D. Buniatyan, T. Macrina, D. Ih, J. Zung, H. Sebastian Seung Deep Learning Improves Template Matching by Normalized Cross Correlation. 2017. arXiv:1705.08593.

[75] Thomas Hossler. Where's Waldo? A Deep Learning approach to Template Matching. 2017. http://cs231n.stanford.edu/reports/2017/pdfs/817.pdf

[76] Blob Analysis. http://docs.adaptive-vision.com/current/studio/machine_vision_guide/BlobAnalysis.html

[77] Computer Vision: Feature detection and matching. Lecture 6. https://courses.cs.washington.edu/courses/cse455/09wi/Lects/lect6.pdf

[78] H. Singh. Vehicle Detection and Tracking using Machine Learning and HOG. https://towardsdatascience.com/vehicle-detection-and-tracking-using-machine-learning-and-hog-f4a8995fc30a

[79] S. Mallick. Histogram of Oriented Gradients. https://www.learnopencv.com/histogram-of-oriented-gradients/

[80] T. B. Dinh, G. Medioni. Co-training Framework of Generative and Discriminative Trackers with Partial Occlusion Handling. 2011.

[81] A. Diba, V. Sharma, R. Stiefelhagen L. Van Gool. Object Discovery By Generative Adversarial & Ranking Networks. 2017. arXiv:1711.08174.

[82] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, H. Wang. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. 2016. arXiv:1607.05781.

[83] Z. Kalal, K. Mikolajczyk, J. Matas. Forward-Backward Error: Automatic Detection of Tracking Failures, *in 20th International Conference on Pattern Recognition (ICPR)*. 2010.

[84] Z. Kalal, K. Mikolajczyk and J. Matas Tracking-Learning-Detection, *in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, no. 7, pages 1409-1422*. 2012.

[85] B. Babenko, M. H. Yang, S. Belongie. Visual tracking with online Multiple Instance Learning, *in 2009 IEEE Conference on Computer Vision and Pattern Recognition, pages 983-990*. 2009. Miami.

[86] J. F. Henriques, R. Caseiro, P. Martins, J. Batista. High-Speed Tracking with Kernelized Correlation Filters. 2014. arXiv:1404.7584

[87] H. T. Nguyen, A. W. M. Smeulders. Fast occluded object tracking by a robust appearance filter, *in IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 26, no. 8, pages 1099-1104*. 2004.

[88] B. C. Russell, A. Torralba, K. P. Murphy, W. T. Freeman. LabelMe: a database and web-based tool for image annotation, *in International Journal of Computer Vision, Volume 77, Issue 1-3, pages 157-173*. 2008.

[89] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, C. Lawrence Zitnick. Microsoft COCO: Common Objects in Context. 2014. arXiv:1405.0312

[90] J. Deng, W. Dong, R. Socher, L. Li, K. Li, L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database, *in 2009 IEEE Conference on Computer Vision and Pattern Recognition*. 2009.

[91] D. Gershgorn. The data that transformed AI research - and possibly the world. https://qz.com/1034972/the-data-that-changed-the-direction-of-ai-research-and-possibly-the-world/

[92] Google Open Images Dataset V4. https://storage.googleapis.com/openimages/web/index.html

[93] J. Huang et al. Speed/accuracy trade-offs for modern convolutional object detectors. 2016. arXiv:1611.10012.

[94] J. Redmon, S. Kumar Divvala, R. B. Girshick, A. Farhadi. You Only Look Once: Unified, Real-Time Object Detection. 2015. arXiv:1506.02640.

[95] J. Redmon, A. Farhadi. YOLO9000: Better, Faster, Stronger. 2016. arXiv:1612.08242.

[96] J. Redmon, A. Farhadi YOLOv3: An Incremental Improvement. 2018. arXiv:1804.02767.

[97] D. Hoiem, Y. Chodpathumwan, Q. Dai. Diagnosing error in object detectors, *in Computer Vision ECCV 2012, pages 340-353*. 2012. Springer.

[98] A. Bewley, Z. Ge, L. Ott, F. Ramos, B. Upcroft. Simple Online and Realtime Tracking, *in 2016 IEEE International Conference on Image Processing (ICIP), pages 3464-3468*. 2016. 10.1109/ICIP.2016.7533003.

[99] N. Wojke, A. Bewley, P. Dietrich. Simple Online and Realtime Tracking with a Deep Association Metric, *in 2017 IEEE International Conference on Image Processing (ICIP), pages 3645-3649*. 2011. 10.1109/ICIP.2017.8296962.

[100] R. Kalman. A New Approach to Linear Filtering and Prediction Problems, *in Journal of Basic Engineering, vol. 82, Series D, pages 35-45*. 1960.

[101] V. Yadav. Kalman filter: Intuition and discrete case derivation. https://towardsdatascience.com/kalman-filter-intuition-and-discrete-case-derivation-2188f789ec3a

[102] S. Srini. The Kalman Filter: An algorithm for making sense of fused sensor insight. https://towardsdatascience.com/kalman-filter-an-algorithm-for-making-sense-from-the-insights-of-various-sensors-fused-together-ddf67597f35e

[103] H. W. Kuhn. The Hungarian Method for the Assignment Problem. 1955.

[104] T. Hoang Trieu. Darkflow. https://github.com/thtrieu/darkflow

[105] Ouail Bendidi. Tracking-with-darfklow. https://github.com/bendidi/Tracking-with-darkflow

[106] G. Ning, Z. Zhang, C. Huang, Z. He, X. Ren, H. Wang. Spatially Supervised Recurrent Convolutional Neural Networks for Visual Object Tracking. 2016

[107] D. Held, S. Thrun, S. Savarese. Learning to Track at 100 FPS with Deep Regression Networks, *in Proceedings of European Conference Computer Vision (ECCV)*. 2016.

[108] D. Gordon, A. Farhadi, D. Fox. Re3: Real-Time Recurrent Regression Networks for Object Tracking. 2017. arXiv:1705.06368.

[109] M. Mueller, N. Smith, B. Ghanem A Benchmark and Simulator for UAV Tracking, *in Proc. of the European Conference on Computer Vision (ECCV)*. 2016.

[110] M. Kristan *et al*. The Visual Object Tracking VOT2016 Challenge Results, *in Hua G., Jégou H. (eds) Computer Vision - ECCV 2016 Workshops*. ECCV 2016. Lecture Notes in Computer Science, vol 9914. Springer, Cham.

[111] M. Kristan *et al*. The Visual Object Tracking VOT2017 Challenge Results, *in 2017 IEEE International Conference on Computer Vision Workshops (ICCVW), pages 1949-1972*. 2017. Venice.

[112] L. Leal-Taixé, A. Milan, I. D. Reid, S. Roth, K. Schindler. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. 2015. arXiv:1504.01942.

[113] A. Milan, L. Leal-Taixé, I. D. Reid, S, Roth, K. Schindler. MOT16: A Benchmark for Multi-Object Tracking. 2016. arXiv:1603.00831.

[114] K. Bernardin, R. Stiefelhagen. Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics, *in EURASIP Journal on Image and Video Processing, Volume 2008, Number 1, pages 246-309*. 2008.

[115] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, C. Tomasi. ID measures for Multi-Target Tracking. `http://vision.cs.duke.edu/DukeMTMC/IDmeasures.html`

[116] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, C. Tomasi. Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking. 2016. arXiv:1609.01775.

[117] L. Leal-Taixé, A. Milan, K. Schindler, D. Cremers, I. D. Reid, S. Roth. Tracking the Trackers: An Analysis of the State of the Art in Multiple Object Tracking. 2017. arXiv:1704.02781.

[118] A. Moudgil, V. Gandhi. Long-Term Visual Object Tracking Benchmark. 2017. arXiv:1712.01358.

[119] Q. Wang. Benchmark Results `https://github.com/foolwood/benchmark_results`

[120] A. Milan, E. Ristani. MOT Challenge benchmark devkit. `https://bitbucket.org/amilan/motchallenge-devkit/`

[121] C. Heindl. Python implementation of MOT Challenge benchmark devkit `https://github.com/cheind/py-motmetrics`

[122] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge, *in International Journal of Computer Vision, volume 88, number 2, pages 303-338*. 2010. `http://host.robots.ox.ac.uk/pascal/VOC/`

[123] M. Everingham, John Winn. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Development Kit. `http://host.robots.ox.ac.uk/pascal/VOC/voc2012/htmldoc/index.html`

[124] Tzutalin. LabelImg. 2015. `https://github.com/tzutalin/labelImg`

[125] CVonline: Image Databases. `http://homepages.inf.ed.ac.uk/rbf/CVonline/Imagedbase.htm#people`

[126] T. Vojir. Short-term visual object tracking in real-time. Institution Department of Cybernetics, Center for Machine Perception. Czech Technical University in Prague. 2017

[127] A. Rosebrock. Intersection over Union (IoU) for object detection. `https://www.pyimagesearch.com/2016/11/07/intersection-over-union-iou-for-object-detection/`

[128] Yuki Tsuji. Yolov2 implementation by Chainer. `https://github.com/yukitsuji/Yolo_v2_chainer`

[129] T. C Arlen. Understanding the mAP Evaluation Metric for Object Detection. `https://medium.com/@timothycarlen/understanding-the-map-evaluation-metric-for-object-detection-a07fe6962cf3`

[130] T. Shah. Measuring Object Detection models-mAP.What is Mean Average Precision? `http://tarangshah.com/blog/2018-01-27/what-is-map-understanding-the-statistic-of-choice-for-comparing-object-detection-models/`

[131] J. Cartucho. mAP (mean Average Precision). https://github.com/Cartucho/mAP

[132] QGroundControl. http://qgroundcontrol.com/

[133] Introduction to TCP-IP. http://www.linux-tutorial.info/modules.php?name=MContent&obj=page&pageid=142

[134] A. Ratner, S. H. Bach, H. R. Ehrenberg, J. Alan Fries, S. Wu, C. Ré. Snorkel: Rapid Training Data Creation with Weak Supervision. 2017. arXiv:1711.10160.

[135] mjpg-streamer. https://github.com/jacksonliam/mjpg-streamer

[136] Raspberry Pi - USB Webcam. https://www.youtube.com/watch?v=JnQh-o9aSD0

Appendix A

The Annotation Process

A group of images do not form a dataset unless they are annotated. This process varies slightly depending on the nature of the Computer Vision algorithm that will take the images. For example, while object classification only needs a label that identifies the class portrayed in the image, for detection bounding boxes along with a class label are needed. In the case of image segmentation, bounding boxes must be either polygons or full boundaries.

In this case the process of annotating simple, rectangular bounding boxes with the class "person" is explained. Particularly, the starting dataset has already some annotations, covering a single subject. The goal is to fully annotate every person of a reasonable size that appears on screen, along with assigning it an id number for that sequence, for multi object tracking performance evaluation. For this purpose, the open source tool LabelImg [124] is used along with a number of scripts written to help with formatting and editing.

A.1 Formats

The UAV123 annotation format is $x1, y1, w, h$, with $x1, y1$ the coordinates of the top-left corner of the bounding box, and w, h the width and height of this box, respectively. Each sequence has its annotations placed on a text file, with the n image of the sequence annotated in the n line of the file. When occlusion happens, *NaN* is used instead of numbers or deserting the line.

```
695,457,15,8
702,390,15,9
704,378,14,6
```

Listado A.1: UAV123 annotation format example

However, the annotation format required by the darflow training script is the PASCAL VOC format. This is also the format used by LabelImg, so the first step is to convert existing annotations from the UAV format, similar to YOLO format, to PASCAL VOC.

```
<annotation>
  <folder>GeneratedData_Train</folder>
  <filename>000001.png</filename>
  <size>
    <width>224</width>
    <height>224</height>
  </size>
  <object>
    <name>21</name>
    <pose>Frontal</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <occluded>0</occluded>
    <bndbox>
```

```
15          <xmin>82</xmin>
16          <xmax>172</xmax>
17          <ymin>88</ymin>
18          <ymax>146</ymax>
19       </bndbox>
20    </object>
21 </annotation>
```

Listado A.2: PASCAL VOC annotation format example

Some attributes are not supported by all implementations. In this case, *pose*, *truncated*, *difficult* and *occluded* will not be used neither by LabelImg nor darkflow, but the attribute *idnumber* will be introduced. This final format will be used by the training process and the evaluation software.

A.2 Workflow

The annotation process is summarized in Figure A.1. First of all, the UAV annotations corresponding to the chosen sequence should be placed in the folder containing the frames, in PASCAL VOC format, and with frame and annotation having matching names. The script *UAV123toPascalVOC.py* will perform this duty.

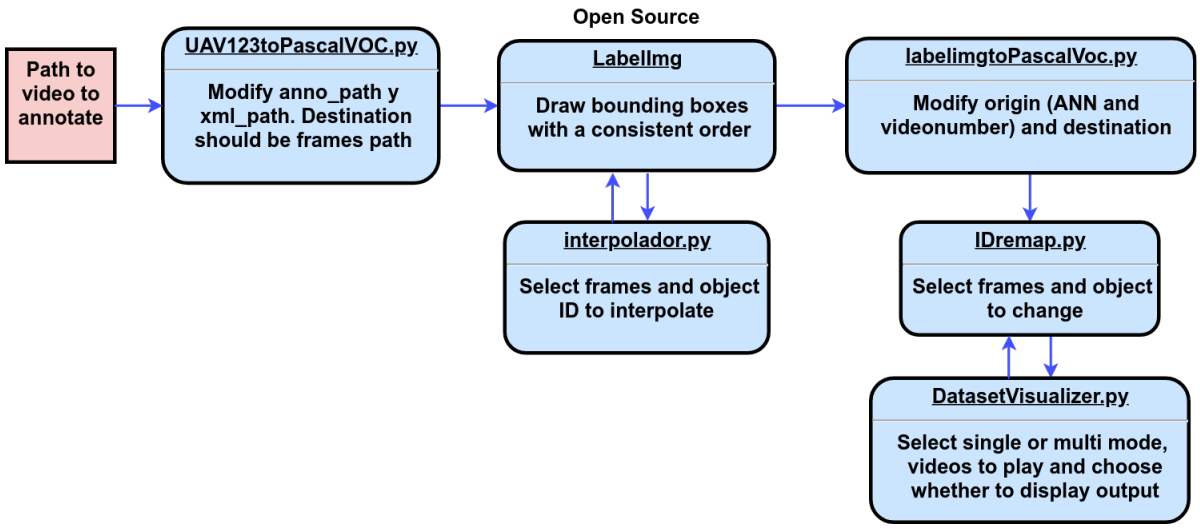


Figure A.1: Annotation process flowchart. Open source software indicated when used, the rest was developed for this purpose.

After that, in the program LabelImg the directory containing the frames and annotations is opened both as Directory and as Save Directory. Then, bounding boxes are drawn and interpolated with *interpolador.py* in a loop, until all people that appear on the sequence have a bounding box. The flowchart shown in Figure A.1 asks that bounding boxes are to be drawn with a consistent order. The explanation is the following: there is no way (as of when this work was done) to assign ID numbers in LabelImg. Therefore, *interpolador.py* will assign IDs to the bounding boxes according to the order they appear on the LabelImg screen. The problem arises when several targets appear and disappear from screen, and do not exist simultaneously.

Let's assume that the target with ID 1 appears during all of the sequence. The second person to appear will be assigned ID 2. When ID 2 leaves the screen, the next person that appears will also be assigned ID, because there are only two bounding boxes on that frame in LabelImg, but this person should be ID 3. This his where a consistent order helps this process. The targets on screen should always be ordered respecting the relative order between them. Every subsequent person that appears on screen should be

annotated the last, and the previous last one should always be annotated previous to the last one, and so on.

This way, it is possible to interpolate frames if during the interpolation the exact same people are on screen, because every time that a person enters or leaves the screen IDs are possibly changed, but the order between them is maintained.

After using *labelimgtoPascalVoc.py*, the annotations are converted to PASCAL VOC format, and should be moved to another directory in order to keep separate instances of the annotation process. At this stage, the annotations have a correct format, but incorrect ID numbers. Now, with *IDremap.py*, ID numbers can be changed between the desired frames. It is now that the consistent order comes in handy. For every segment of the sequence where the same targets are on screen, ID numbers can easily be rearranged with this script. Starting from the last one (greatest ID number) so as to avoid mixing annotations of changed and unchanged IDs, IDs can be remapped. It may be useful to use both LabelImg (for precise frame numbers) and *DatasetVisualizer.py* (for absolute ID number and checking) during this process.

A.3 Annotation Philosophy

Bounding boxes should enclose the target as perfectly as possible, but there are two tradeoffs here. One is bounding box accuracy vs time required, and the other is annotating train-oriented or tracking-oriented.

In the first case, the more precise the bounding box is, more time will be required to finely adjust it, especially during abrupt camera movement or non-linear trajectories on screen, which are very difficult to interpolate (requiring interpolations of less than 5 or 10 frames, while linear motion can be successfully interpolated up to 50 or even 100 frames). Nevertheless, when the apparent motion of the bounding box in the screen (result of the simultaneous movement of camera and target) is near linear, it can be comfortably interpolated and may be only necessary to manually annotate around 5% or less of the frames (Figure A.2).

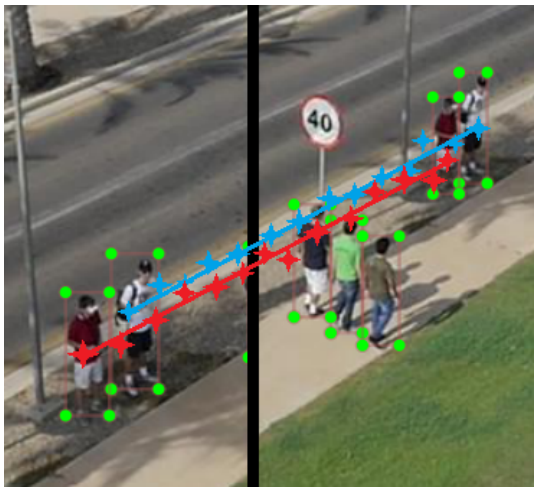


Figure A.2: Image composed of two frames of the same sequence, taken at different times. When apparent bounding box motion is near linear, interpolation is applicable.

There is no quality threshold, so it is for the annotator to decide how much time to invest, taking into account the diminishing returns of adjusting bounding boxes pixel by pixel in every frame.

The second tradeoff is a more subtle design choice. If a partial occlusion happens and is significant enough that a person is not easily recognizable, it should not be fed into the training process because it

is not a person, and will increase detection error. Therefore, it should not be annotated. However, it is expected of a tracker to be able to follow the target while partially occluded, specially when considering online learning trackers. There is a region where the target is no hardly recognizable as a person by just its current appearance, but given its previous motion it should be identifiable as the target.

Here lies the dichotomy. If annotation is intended for training, only clearly visible people should be annotated (at least half the body is visible). Instead, if tracking performance evaluation is the goal, annotation should cover all frames where the target is either sufficiently visible and given its trajectory tracking is expected. The answer is frequently contradictory and, since datasets are not usually conceived for a single purpose, no answer is the correct one.

Figure A.3: Two frames from the UAV123 dataset where partial occlusion happens. Are the people on this frames clearly, visible people? Would you expect of a tracker to be able to follow the target during these occlusions?

For this work, it has been annotated as person most of the bounding boxes containing either approximately a third of the body at least or easily recognizable features such as the head when accompanied by proper, smooth trajectories. Of course, this is rather subjective. However, it could be argued that the influence of these 'spurious' annotations, where little of a person is recognizable, do small harm to the detector's performance (no more than imprecise bounding boxes or people not annotated because of its small size due to being far from the camera), while it is precisely during partial occlusion where a tracker is most wanted to keep the target, and thus these situations need to be evaluated. Since it is hard to approach this problem quantitatively, and having two separate annotations for training and evaluating largely falls out of the question because of practical considerations, this is poised to remain part of the annotators' preferences.