EVALUATION OF DEEP LEARNING BASED MULTIPLE OBJECT TRACKERS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

OMAR MOURED

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONIC ENGINEERING

SEPTEMBER 2020

Approval of the thesis:

EVALUATION OF DEEP LEARNING BASED MULTIPLE OBJECT TRACKERS

submitted by **OMAR MOURED** in partial fulfillment of the requirements for the degree of **Master of Science in Electrical and Electronic Engineering, Middle East Technical University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences** _____

Prof. Dr. İlkay Ulusoy
Head of the Department, **Electrical and Electronics Eng** _____

Prof. Dr. Gözde Bozdağı Akar
Supervisor, **Electrical and Electronics Eng, METU** _____

Examining Committee Members:

Prof. Dr. İlkay Ulusoy
Electrical and Electronics Eng, METU _____

Prof. Dr. Gözde Bozdağı Akar
Electrical and Electronics Eng, METU _____

Prof. Dr. Aydın Alatan
Electrical and Electronics Eng, METU _____

Prof. Dr. Alptekin Temizel
Graduate School of Informatics, METU _____

Assoc. Prof. Dr. Erkut Erdem
Computer Eng, Hacettepe University _____

Date: 16.09.2020

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name : Omar Moured

Signature :

ABSTRACT

EVALUATION OF DEEP LEARNING BASED MULTIPLE OBJECT TRACKERS

Moured, Omar
Master of Science, Electrical and Electronic Engineering
Supervisor : Prof. Dr. Gözde Bozdağı Akar

September 2020, 117 pages

Multiple object tracking (MOT) is a significant problem in the computer vision community due to its applications, including but not limited to, surveillance and emerging autonomous vehicles. The difficulties of this problem lie in several challenges, such as frequent occlusion, interaction, intra-class variations, in-and-out objects, etc. Recently, deep learning MOT methods confront these challenges effectively. State-of-the-art deep learning (DL) trackers pipeline consists of two stages, i.e., appearance handling, which includes object detection and feature extraction, and grouping step, which performs affinity computation and data association. One of the main concerns of this thesis is to investigate how DL was employed in each one of these stages. In addition to that, we have experimented with different performance-enhancing methods, the currently top online tracker on the MOTChallenge dataset. Based on the investigation and experiments, we will identify and discuss the significant shortcomings of the current frameworks, providing possible ways to improve it.

Keywords: Multiple Object Tracking, Deep Learning, Online Tracker

ÖZ

EVALUATION OF DEEP LEARNING BASED MULTIPLE OBJECT TRACKERS

Moured, Omar
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği
Tez Yöneticisi: Prof. Dr. Gözde Bozdağı Akar

Eylül 2020, 117 sayfa

Çoklu nesne takibi (ÇNT/MOT), bilgisayarlı görme alanında, aralarında gözetleme ve yeni geliştirilen otonom araç uygulamalarının olduğu ancak bunlarla sınırlı olmayan çok geniş çeşitlilikte uygulamalarda kullanılan oldukça önemli bir konudur. Bu konudaki zorluklar, sıklıkla oluşan kapanmalar, etkileşimler, sınıf içi değişimler, giren-çıkan nesneler vb. şeklinde sıralanabilir. Yakın geçmişte, derin öğrenme ÇNT (MOT) yöntemleri verimli bir şekilde bu zorlukların üstesinden gelmiştir. En gelişkin derin öğrenme (DÖ/DL) takipçilerinin sıralı düzeni, nesne algılama ve detay çıkarma yöntemlerini içeren görünüş düzenleme ve benzeşim hesaplamaları ile veri ilişkilendirmenin yapıldığı gruplandırma adımı gibi iki aşamadan oluşmaktadır. Bu tezin başlıca kaygılarından biri, DÖ (DL)'nün bu aşamaların her birinde nasıl kullanıldığını araştırmaktır. Buna ek olarak, MOTChallenge veri setinde bulunan ve mevcut durumdaki en iyi çevrimiçi takipçi, birbirinden farklı performans iyileştirici yöntemler kullanılarak deneylere tabi tutulmuştur. Araştırmalar ve deneylere istinaden, güncel sistemlerin göze çarpan eksiklikleri, bu eksikliklerin geliştirilmesi için muhtemel yöntemler belirtilerek tanımlanıp tartışılacaktır.

Anahtar Kelimeler: Çoklu Nesne Takibi, Derin Öğrenme, Çevrimiçi Takipçi

To my beloved family and beautiful country …

ACKNOWLEDGMENTS

I would like to express my very special gratitude to the backbone of this work, my supervisor Prof. Dr. Gözde Bozdağı Akar, for her ever-lasting guidance and understanding. This thesis would not be possible without her support that has started from the first day of my MSc study. Along with her easygoing attitude, she kindly provided access to reliable computational devices to conduct the experiments. Thanks to her, I was able to explore different areas of machine learning.

I am also profoundly grateful to my family. I know my words are insufficient with everything you have done for me. I am thankful to my Father Ahmed, my Mother Souhila, and my Brothers Taki and Salah for all the support and inspiration they have provided throughout my whole life.

TABLE OF CONTENTS

LIST OF TABLES

TABLES

LIST OF FIGURES

FIGURES

CHAPTER 1

INTRODUCTION

Object tracking denotes the problem of utilizing measurements acquired from sensors (sonar, lidar, camera, infrared sensor, etc..) to collect information about objects. The typical aim of tracking is to determine the states of the scene objects such as velocity, position, or features. Generally speaking, the tracking tasks can be divided into two categories according to the number of targets being tracked—namely, Single Object Tracking (SOT) or Multiple Object Tracking (MOT).

MOT is concerned with forming a trajectory for multiple objects of interest (from predefined classes) simultaneously. Over the last decade, It headlined tech news due to its potential in many disciplines, ranging from surveillance to biology. Surveillance systems are deployed in cities to identify the individuals or vehicles of interest for traffic monitoring. MOT is used in sports events to track and analyze the players accurately [1]. It is also utilized to automate the tracking of animal movement and behavior [2]. In the biomedical field, MOT is used to analyze the proliferative response of stem cells [3] for drug discovery and stem cell manufacturing [4].

SOT, on the other hand, is concerned with tracking a target of interest in which its appearance is known a prior. The critical difference is that MOT needs to identify the relationships between all available targets in the scene.

Due to recent progress in the Deep Learning (DL) field, especially the convolutional neural network (CNN) for object detection, The MOT problem has also achieved great success. Most of MOT state-of-the-art approaches perform much better compared to traditional methods that involve manual feature extraction. The shift of attention towards DL in the MOT task started in 2012 when Krizhevsky et al. [5] won ILSVRC (ImageNet Large-Scale Visual Recognition Challange). They managed to reduce image classification Top-1 error by 9.6%, which was surprising at that time. A few years later, CNN was adopted to other tasks such as object

detection, semantic segmentation, and image captioning, providing more robust tools for MOT approaches. Despite the difficulties and challenges present in MOT, deep learning-based methods have stood out in the leaderboard of all available MOT benchmarks. A vital part of MOT is to perceive unique and distinct features of present objects in a frame throughout their lifespan in the video. DL in MOT has traditionally been done in a tracking-by-detection approach, relying on an object detection models then performing association step. Recent research has introduced end-to-end DL tracking ways that can form a unique dynamic state for every object in the scene. We call trackers that utilize future frames when handling the current frame as offline or batch trackers. Online or causal tracking methods only rely on past and present information; they process the video frames sequentially in a step-wise manner, which makes them identical to real-life scenarios (e.x. video stream). In this thesis, we pipelined the MOT task to 2 stages, and we will discuss how DL is utilized to enhance the overall performance in each one. Motivated by this, we propose an improved DL pedestrian tracker, which is inspired by the current top online tracker in the MOTChallenge leaderboard.

1.1 Problem Statement and Motivation

Object detection and MOT are two distinct machine vision problems. While they are distinct problems, they are strongly related. A typical scenario that uses object detection, or sometimes called static object detection, is depicted in Figure 1.1. In this scenario, the object detection model outputs bounding boxes that localize the players. One can define object detection task as the problem of estimating the classes and locations of objects in a given image. However, in real-life scenarios, one may be interested in evaluating the three players with orange shorts throughout the run and maybe estimate their performance in the upcoming races. MOT does this task. MOT method is not just localizing the objects but also assign them to the correct trajectories, which require the utilization of temporal information too.

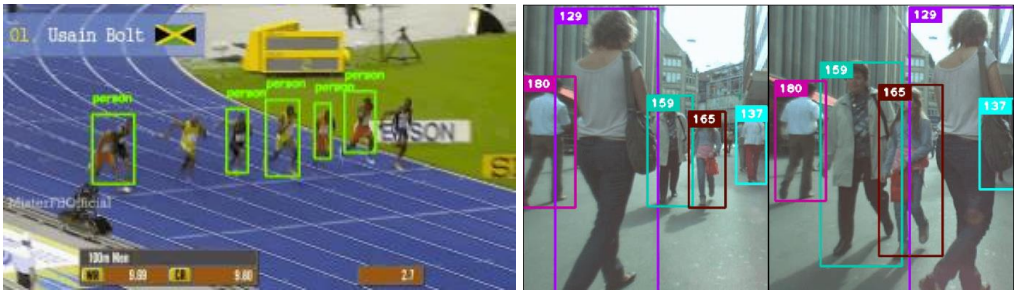


Figure 1.1. On the left, object detection model outputs. On the right MOT outputs. [6], [7]

One may argue that MOT can be achieved with classification models. That is, for every unique object, we have a classification model to predict if the bounding box belongs to the same object or not. However, this approach is not realistic because it requires prior knowledge of all objects presented in the video with the labeled dataset (maybe from other videos for the same object) to train the classifiers. Moreover, different classifiers may conflict and claim to include the same bounding box.

For this reason, it is required to learn robust features that are distinct for every object belong to the same class (e.g., pedestrian, vehicle) and are the same under different appearance changes (throughout temporal dimension). Many challenges arise with MOT compared to static object detection. Different types of challenges are exemplified in Figure 1.2. Although some of these challenges may be faced by object detection models, the extension of the temporal dimension makes it more robust.

Figure 1.2. Various MOT challenges. Frames were collected from the MOT20 benchmark [8]. (a) represent dense 200+ pedestrian scenes with intra-class variations. (b) Re-entering objects, which raise the problem of re-identification. (c) Partially and fully occluded objects. (d) Sudden illumination change. For dynamic cameras, viewpoint change is also present.

1.2 Contributions

This thesis has two main contributions: An evaluation of the state-of-the-art DL based MOT methods. More specifically, the top published online trackers in MOTChalenge. From the evaluation process, the bottlenecks are emphasized, and novel solutions are proposed—precisely, a calibration technique for the Faster R-CNN detection model and a high-speed LSTM based association network.

1.3 Outline

As mentioned earlier, we prefer to pipeline DL based MOT methods into two steps. An appearance handling stage and grouping step. As shown in Figure 1.3, each step is a combination of two other stages.

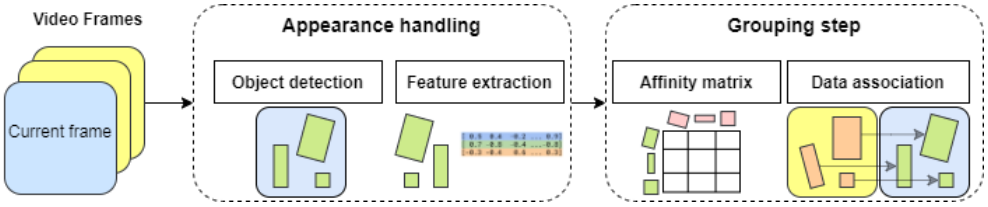


Figure 1.3. DL based trackers pipeline.

The reason we didn't pipeline them to 4 steps is that some approaches have an end-to-end appearance handling system. That is, the same model is used to localize and extract features in one step, while a few others separated object detection step from feature extraction. The same applies to the grouping stage. The rest of the thesis is organized as follows:

- **Chapter 2** presents a theoretical background in deep learning. This chapter covers the essential tools to understand the rest of the thesis.
- **Chapter 3** will cover the datasets and the evaluation metrics we will follow to perform a fair comparison between the different methods.
- **Chapter 4** present the first step of DL based MOT methods, which is appearance handling. It will cover the tricks that recent trackers use to enhance object detection step. It will also describe the different approaches to extract tracklet (object trajectory) features.
- **Chapter 5** will be the continuation of the tracking pipeline – that is, the grouping step. It illustrates how to form an affinity matrix with DL between tracklets and new detections for association task.
- **Chapter 6** will cover the numerical evaluation with the bottlenecks. The chapter will also discuss the baseline approach we targeted in our proposal.
- **Chapter 7** will discuss how did we attack the problems mentioned in chapter 6 and our novel solutions.
- **Chapter 8** concludes the thesis.

CHAPTER 2

A BRIEF BACKGROUND ON DEEP LEARNING

Until today, Deep Learning is still an ambiguous term, as it has gone through many changes throughout the years. One can find a modern definition at Goodfellow's book "*Deep Learning*" [9]. There, it is defined as a tool that allows computers to learn from experience and understand the world in terms of the hierarchy of concepts. In this thesis, we will follow a narrower definition proposed by Skansi's book [10]. There, it refers to DL as a subfield of machine learning.

At the beginning of this chapter, we will discuss a brief overview of machine learning. Then, vanilla (basic) neural networks are discussed, and how are they trained. At the end of this chapter, we will focus on convolutional and recurrent neural networks, which are the building blocks of object detection and reidentification models.

2.1 Machine Learning overview

Machine learning (ML) is a subfield of Artificial Intelligence (AI). ML models use a set of statistical tools to estimate complicated functions based on sample data. Figure 2.1 shows one application where ML is used to classify hand-written numbers. Although it is a trivial task for a human to read numbers, for computers, these digits are a punch of pixel values spread in an XY-plane. Hence, machines need an algorithm defined by the user to tell them how to handle and process inputs. Since it is impossible to write an algorithm for evert input, ML uses statistical models to build knowledge about the dataset.

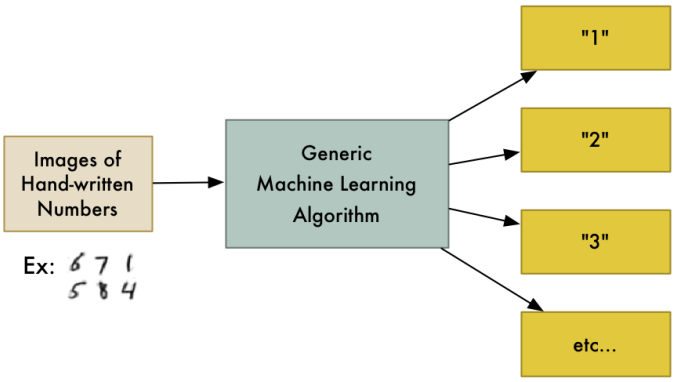


Figure 2.1. Classifying hand-written numbers with machine learning [11].

Machine learning can be divided into three main approaches, as shown in Figure 2.2. They mainly differ in the form of the input data.

2.1.1 Supervised Learning

In supervised learning, we use labeled data to train the machine learning model. We pass the input to the model, and we compare its output with the correct label using predefined criteria by the user called loss function. A simple supervised learning training cycle is presented in Figure 2.5. The aim here is to map the dataset to the correct labels with minimum loss. Object detection and tracking are examples of supervised learning. Most of the state-of-the-art tracking methods make use of supervised learning for the reason that all of the DL object detection models are trained with this approach. However, few recent publications have used the other two learning approaches with decent performance, such as [12], [13].

The two most performed tasks with supervised learning are:

1. **Classification:** is the task of approximating a mapping function $f(.)$ From input variables X to discrete output Y. The output variables are often called labels or categories. The mapping function predicts the class or category for a given observation. Given an input dataset of hand-written digits, they are predicting which image corresponds to which digit is a classification task.

2. **Regression:** regression, on the other hand, is the task of approximating a mapping function $f(.)$ From input variables X to a continuous output variable Y. Y often represent quantities, such as amounts and sizes. Predicting the coordinates of digits in a license plate is a regression task.

2.1.2 Unsupervised Learning

In contrast to supervised learning, the unsupervised learning approach uses unlabeled data for training. The goal here is to find structure inside the given input; it is used, for example, in clustering and dimensionality reduction.

2.1.3 Reinforcement Learning

Reinforcement learning makes use of a feedback loop that provides new data in response to a set of the decision made by the model. Unlike supervised learning, the model is not provided with correct labels but will be rewarded if the right choice is made or penalized otherwise (using predefined rules). The goal here is to maximize total reward by forming a decision policy.

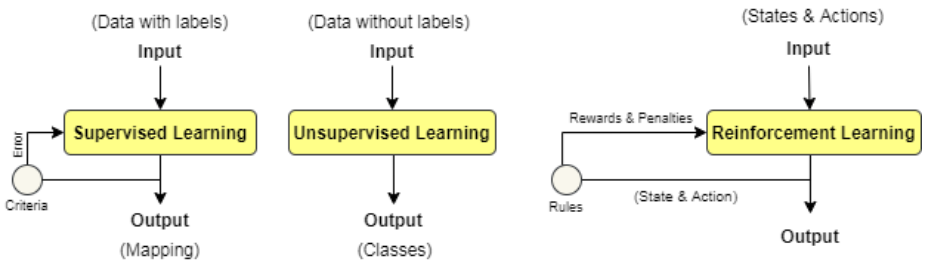


Figure 2.2. Three different types of machine learnings [14].

2.2 Data Preprocessing

The learning process of machine learning models is highly influenced by the size of the datasets and the data-preprocessing techniques used [15]. Before feeding training data to the network, it is usually preprocessed. The widely used form for preprocessing is first by subtracting the mean from the dataset. It has the geometric interpretation of centering the samples of data around the origin in every dimension. For RGB images, all samples are added, then the mean is computed for every channel then the corresponding mean is subtracted from every pixel in that channel. Another technique for dealing with very different feature ranges is to rescale the samples across every dimension. Equations 2.1-2.3 shows some other used methods for preprocessing. $\boldsymbol{x_i}$ represents the $\boldsymbol{i^{th}}$ channel of the input sample $\boldsymbol{x}$, and $\boldsymbol{X_i}$ is the i^{th} channel of the whole dataset $\boldsymbol{X}$ **(**after summation). μ and σ, are the mean and standard devidation of the input sample respectively.

$$Min - Max\ Rescaling \quad \boldsymbol{x_i} \coloneqq \frac{\boldsymbol{x_i} - min(\boldsymbol{X_i})}{max(\boldsymbol{X_i}) - min(\boldsymbol{X_i})} \tag{2.1}$$

$$Normalization \quad \boldsymbol{x_i} \coloneqq \frac{\boldsymbol{x_i}}{\|\boldsymbol{x_i}\|} \tag{2.2}$$

$$Standardization \quad \boldsymbol{x_i} \coloneqq \frac{\boldsymbol{x_i} - \mu_{\boldsymbol{x}}}{\sigma_{\boldsymbol{x}}} \tag{2.3}$$

2.3 Artificial Neural Network

A neural network is a classifier imitating how the human brain works. Neurons in a human brain are cells that can transmit information to other nerve cells [16]. Each cell collects inputs from other neurons that are connected to forming a very complex network of signal transmission.

Human brain neurons are mimicked by perceptrons (basic units) in ANN. As depicted in Figure 2.3, the perceptron can take several inputs $\{x_1, x_2, .., x_n\}$ that might represent pixels in an image. These inputs are multiplied with the associated

weight ($w_n$in the nth connection). The weighted sum is passed to an activation function $\varphi(.)$, and if the output of the activation exceeds a threshold (usually zero), the perceptron is activated and will send an output. The output signal y depends on the activation function used and is often a binary signal, either -1 and 1 or 0, and 1. b represent the bias which shift the decision boundary away from the origin. The mathematical formulation of the perceptron is presented in Equation (2.4). $\boldsymbol{w}.\boldsymbol{x}$ represent dot product between the weights and inputs vectors.

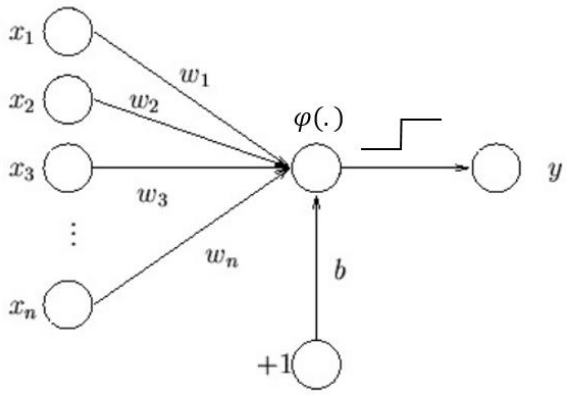


Figure 2.3. A perceptron with n inputs [9].

$$y(\boldsymbol{x}) = \begin{cases} 1, & if\ \varphi(\boldsymbol{w}.\boldsymbol{x} + b) \geq\ threshold \\ -1, & otherwise \end{cases} \tag{2.4}$$

Stacking and concatenating perceptrons will form an ANN, such as the one in Figure 2.4. multilayer NN is capable of solving nonlinear mapping such as the XOR problem. Usually, a NN with more than two layers is referred to as deep NN. A classifier must be able to learn from examples, and in an ANN, the most remarkable contribution was perhaps the learning algorithm. There are several ways of doing this. However, all of them involve initializing the weights and then passing an example through the network. The error made by the model is calculated with a loss function and fed backward through a process called "backpropagation" to update the weights. The last two steps are usually referred to as an optimization process. In the upcoming sections, we will briefly discuss these steps.

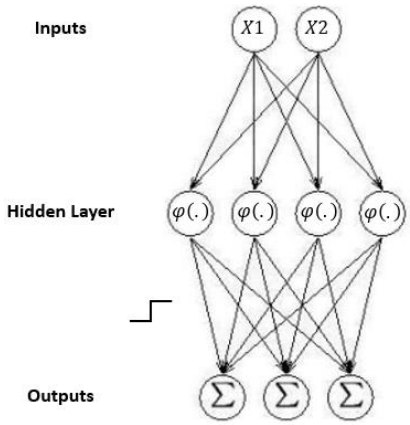


Figure 2.4. A graphical representation of ANN with one hidden layer [9].

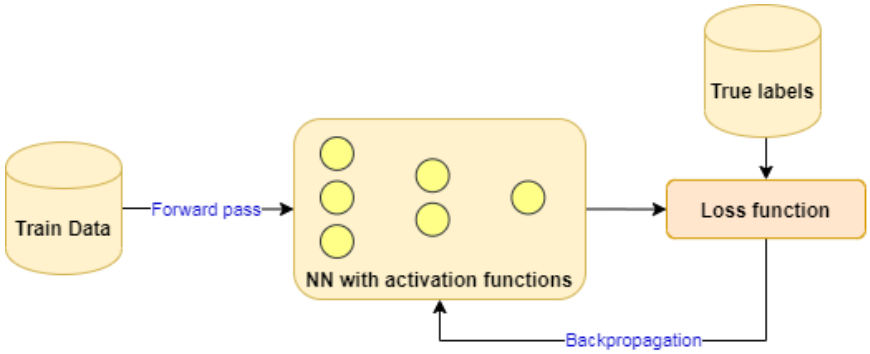


Figure 2.5. Supervised learning based NN pipeline.

2.3.1 Weights Initialization

Weights are specified by the backpropagation algorithm, which means that it is based on finding the surface of an error (error as ANN weight function). NNs learning efficiency depends much on the distribution of the initial weights. Moreover, the ability of convergence and the speed of that is influenced by the initialization. Weights can't directly be initialized to zeros. Passing any sample data from the training set will result in zero output. Hence, all the weights will get the same gradient (i.e., no source of asymmetry). In addition to that, initializing weights too small or big might cause gradient vanishing or explosion, respectively. These two terms were introduced by Bengio et al. in 1994 [17]. He defined the exploding

gradient problem as the significant increase in the norm of the gradient during training, which makes the long term components grow exponentially more than the short term ones. The vanishing gradient problem refers to the opposite behavior when the long term components go exponentially fast to zero, making it impossible for the NN model to learn the correlation between distant samples. In this section, we will cover the most commonly used heuristics for weight initialization. However, this is by no means a formal and in-depth view of the topic. Interested readers can find more information in [18]–[20].

2.3.1.1 Xavier Initialization

X. Glorot and Y. Bengio proposed the Xavier initialization method in 2010 [20]. The goal was to break the symmetry problem and to have a stable backpropagation signal in deep NN. The first step of Xavier initialization is to sample the weights $\boldsymbol{w}$ from a uniform distribution, preferably with a range $[-0.5,0.5]$ (in fixed intervals, as in the original paper), or gaussian distribution $\mathcal{N}(\mu = 0, \sigma^2)$ with small variance σ^2 (usually 0.1 or 0.01). Second, scale the weights propotional to the number of inputs to the layer. In particular, scaling is done as follows:

$$\boldsymbol{w}^i \coloneqq \boldsymbol{w}^i . \sqrt{\frac{1}{m^{(i-1)}}} \tag{2.5}$$

where $\boldsymbol{w^i}$ Represent the sampled weights of the ith layer, and m is the number of input features connected to the neuron. For the first hidden layer, m will be the number of input channels; for the second hidden layer, that is the number of units in the 1st hidden layer, etc. If biases are not initialized to zeros, they should be included in the scaling too. The scaler one and the variable m may slightly vary depending on the activation function followed by the neuron. Up to date, this is the default method for weights initializing in DL frameworks (such as Tensorflow and PyTorch). The rationale behind this scaling is well explained in the original paper.

2.3.1.2 Layer-Sequential Unit Variance Initialization (LSUV)

The LSUV initialization is proposed by D. Mishkin and J. Matas in 2015 [19]. Unlike the Xavior method, LSUV depends on input data to fine-tune the weights. At first, the weights $\boldsymbol{W}$ are sampled from unit variance gaussian distribtion $\mathcal{N}(\mu = 0, \sigma^2 = 1)$. Second, decompose them with an orthonormal basis such as SVD-decomposition and replace the weights with one of the components. Finally, rescale the weights iteratively to have a variance of one. That is, for every layer i, forward pass a minibatch of training data and find the variance of the output blob O. if the computed variance is not 1 (or maximum number of iteration is not reached), scale the weights by equation (2.6). In the original paper, a small deviation from unit variance is accepted.

$$\boldsymbol{w}^i \coloneqq \frac{\boldsymbol{w}^i}{\sqrt{Var(O^i)}} \tag{2.6}$$

2.3.1.3 Transfer Learning

We may have a classification task in one domain, but we only have few training data in another domain of interest, where the latter data may be in a different distribution or different feature space. In such a case, transfer learning would significantly save labeling effort and learning time. In practice, it is common to use weights of pre-trained models for performing a similar task on a different problem. Transfer learning usually takes the following forms:

1. **Frozen feature extractors:** In this form, the weights of a pre-trained model from the first layers are utilized in another NN. Along with that, new hidden layers are added on the top initialized with any of the beforementioned methods. Trained weights are used as a feature extraction of the input samples and are frozen during training. Meaning, they will not be updated by the backpropagation process.

2. **Fine-tuned weights:** In contrast to the previous point, all of the preprinted weights are utilized and used as an initial position for the new model. The training is done as a whole. Meaning, all the weights are updated.

In transfer learning, The convergence and training time of the new model relies on the similarity of the training dataset to the original one. One of the disadvantages of transfer learning is that the architecture of the new model will depend on the shape of the pre-trained weights. For example, if the first approach is used, the input shape should exactly match the preprinted one, which might require scaling the dataset and lose some of its characteristics. The paper [21] provides a good insight into this field.

2.3.2 Activation Functions

Activation functions (AF), or referred o as transfer functions, are the core elements of the neurons. They determine whether the neuron should be activated (fired) or not. The common problem for most NN models is how the gradient flows within the network, owing to the fact that some gradients are fast in one direction and slow or zero in others. Turian et al. [22] highlighted three key areas to improve learning performance. One of them is to engineer better features by proper choice of AFs. Certainly, neglecting the use of AF and perform a linear mapping from inputs $\boldsymbol{x}$ to output y in Figure 2.3 will allow the model to learn linear tasks only. That is becuase the dot product is a linear operation. To approximate non-linear relations from input features to output labels, non-linear activation functions are utilized. A vital property of the non-linear AFs is that they are differentiable; else, they cannot work during the backpropagation of the deep neural networks. In this section, we will review common AFs Used in the state-of-the-art NNs.

2.3.2.1 The Sigmoid function

Sigmoid function has an "S" shaped curve Figure 2.6(a). It is a real-valued, differentiable, and strictly increasing function. Takes real input and output value in the interval $[0,1]$. The mathematical formulation of the sigmoid function σ Is shown in equation (2.7), where x is the AF input (scalar). Sigmoid can be interpreted as a probability function describing the chance that a neuron fires in a given time interval. For this reason, it is usually used at the classification layer of NNs. For multiple classes task, a similar function Softmax is applied.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.7}$$

2.3.2.2 Rectified Linear Unit (ReLU) function

ReLU is a non-linear activation function that was introduced in 2000 [23]. In 2011, it was demonstrated to improve the training of deep neural networks further. ReLU works by thresholding values at 0, as in equation (2.8), where x is the AF input (scalar). Simply put, It outputs 0 if $input < 0$ and linear function when $input \geq 0$ (refer to Figure 2.6(c)) for visualization). It is usually the first choice to use in hidden layers because it does not suffer from vanishing gradient as in sigmoid function.

$$f(x) = max(0, x) \tag{2.8}$$

2.3.2.3 Hyperbolic Tangent Function (Tanh)

The hyperbolic tangent function in equation (2.9) is a smoother [24], zero-centered activation function whose range lies between -1 to 1. It is preferred in the hardware implementation of NN since it requires a small number of logic elements compared to other AFs [25]. x in the above equation is the AF input (scalar).

$$h(x) = \frac{e^{2x} - 1}{e^{2x} + 1} \tag{2.9}$$

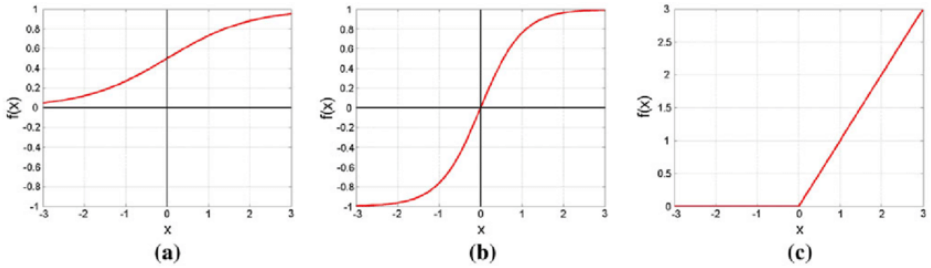


Figure 2.6. (a) Sigmoid function, (b) Tanh function, (c) ReLU function.

2.3.3 Loss Functions

A good teacher not only provides his/her student with high-quality education materials but also set up appropriate learning objectives considering different situations of students. When it comes to artificial intelligence, treating NN models as students, loss function, or referred to as cost function, is the teacher that seeks to minimize the objective function loss [9]. That is, to find the candidate solution that has the lowest score possible. Loss functions are also described to be "the effective drivers of the network's learning" [26]. In this section, we will review some of the commonly used classification losses for the object detection task.

2.3.3.1 $\ell 2$ Loss

This choice of the cost function generally defaults to the euclidean norm. Given the many desirable properties this norm has, it is significantly affected by the image quality (impact of noise) and resolution (image size) [26], which is a severe problem for object detection models. Mean Squared Error (MSE) equation (2.10) is only concerned with the average magnitude error irrespective of their direction. n represent the total number of passed samples. y_i is the NN's output for the ith sample and $\hat{y}_i$ is the desired label. In the context of object detection, labels refer to the class of the bounding box. For coordinates prediction smooth $\boldsymbol{\ell 1}$ norm is used.

$$C(\boldsymbol{y}, \hat{\boldsymbol{y}}) = \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{n} \tag{2.10}$$

2.3.3.2 The Cross-Entropy Loss

The cross-entropy is a measure from the field of information theory. It is used to calculate the difference between two probability distributions based on the entropy. Using cross-entropy instead of $\ell 2$ leads to faster training as well as improved generalization [27]. It is usually used for models that output a probability value between 0 and 1. The Faster R-CNN [28] object detection model uses cross-entropy as a classification loss. Equation (2.11) shows the mathematical formulation of the cross-entropy loss.

$$C(\boldsymbol{p}) = -\sum_{c=1}^{N} y_{(o,c)} * log\left(p_{(o,c)}\right) \tag{2.11}$$

N represents the total number of classes. $\boldsymbol{p}$ is an N dimensional vector generated by the model, and it consists of the probabilities of input being one of the targeted classes c. y is a binary indicator, and it is 1 if the predicted class c is the correct classification for the observation o, i.e., we only penalize the prediction that corresponding to the desired output class. $p_{(o,c)}$ Is the prediction probability for observation o of class c.

2.3.3.3 The Hinge Loss

The Hinge loss, or referred to as $\ell 2$ regularized loss, is used for maximum-margin classification, most notably for supported vector machine (SVMs). In contrast to $\ell 2$ and cross-entropy loss, Hinge penalizes predictions not only when they are incorrect, but even when they are correct but not confident. Hinge loss is also easy to compute since its gradient is zero a lot of the time, which means no weights update required. For an intended output class c and prediction vector $\boldsymbol{y}$, the Hinge loss for an input $\boldsymbol{x}$ can be computed with the equation (2.12). $\hat{\boldsymbol{y}}$ represent the output vector with probabilities for M number of classes. $y(\boldsymbol{x})$ is the ground-truth label index. Simply

put, the Hinge loss forces the desired class probability to be favorable and more abundant than other class predictions.

$$C(\hat{\boldsymbol{y}}) = \sum_{i \neq y(\boldsymbol{x})}^{M} max\left(0, \hat{y}_i - \hat{y}_{y(\boldsymbol{x})} + 1\right) \tag{2.12}$$

2.3.4 Backpropagation

Backpropagation is a widely used method to calculate the gradient of loss functions with respect to NN weights (using the chain rule) for each input sample. There are many advantages that backpropagation posses, here are a couple:

1. Backpropagation calculates the gradient in a single pass for every single parameter (in contrast to the finite difference method).
2. It measures the exact gradient instead of the approximated one.
3. Suitable for dynamic programming. It can iterate backward to avoid redundant calculations of the gradient. i.e., the advance layer gradient can be used to calculate the next one.

Considering the neural network in Figure 2.7, we will apply backpropagation to find the gradient of each layer. After the gradient is computed, we will review optimization methods that are used to iteratively adjust the parameters in the reverse direction of the gradient to minimize the loss function.

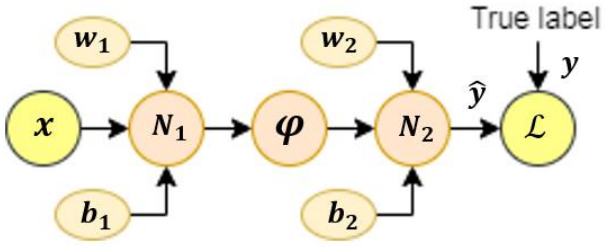


Figure 2.7. Simple two-layer NN with input $\boldsymbol{x}$, activation φ, loss $\mathcal{L}$ functions.

First, the forward pass relationships are found as in equations 2.13 – 2.15

$$N_1 = w_1 \cdot x + b1 \tag{2.13}$$

$$\hat{y} = N_2 = w_2 \cdot \varphi(N_1) + b_2 \tag{2.14}$$

$$\mathcal{L}(y, \hat{y}) = \frac{1}{2} \cdot \|y - \hat{y}\|^2 \tag{2.15}$$

Then the backward pass is computed by passing messages (gradient) in a step-wise manner. The aim is to find equations 2.16, 2.17, 2.21, 2.22 to update w_i and b_i.

$$\frac{\partial \mathcal{L}}{\partial \hat{y}} = (y - \hat{y}) \tag{2.16}$$

$$\frac{\partial \mathcal{L}}{\partial w_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial w_2} = (y - \hat{y}) \cdot \varphi(N_1)^T \tag{2.17}$$

$$\frac{\partial \mathcal{L}}{\partial b_2} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial b_2} = (y - \hat{y}) \tag{2.18}$$

$$\frac{\partial \mathcal{L}}{\partial \varphi} = \frac{\partial \mathcal{L}}{\partial \hat{y}} \cdot \frac{\partial \hat{y}}{\partial \varphi} = w_2^T \cdot (y - \hat{y}) \tag{2.19}$$

$$\frac{\partial \mathcal{L}}{\partial N_1} = \frac{\partial \mathcal{L}}{\partial \varphi} \cdot \frac{\partial \varphi}{\partial N_1} = w_2^T \cdot (y - \hat{y}) \cdot \varphi'(N_1) \tag{2.20}$$

$$\frac{\partial \mathcal{L}}{\partial w_1} = \frac{\partial \mathcal{L}}{\partial N_1} \cdot \frac{\partial N_1}{\partial w_1} = w_2^T \cdot (y - \hat{y}) \cdot \varphi'(N_1) \cdot x^T \tag{2.21}$$

$$\frac{\partial \mathcal{L}}{\partial b_1} = \frac{\partial \mathcal{L}}{\partial N_1} \cdot \frac{\partial N_1}{\partial b_1} = w_2^T \cdot (y - \hat{y}) \cdot \varphi'(N_1) \tag{2.22}$$

2.3.5 Optimization

The aim of calculating the gradient is to minimize the loss function. Optimization is done to speed up the minimization process. In this section, we will discuss the most utilized optimization methods to train NNs.

2.3.5.1 Stochastic Gradient Descent (SGD) Optimizer

Gradient descent (GD) in equation (2.23) is a popular optimization technique that iteratively utilizes the gradient to converge to a local minimum.

$$w_i := w_i - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_i} \tag{2.23}$$

In GD, the weights are updated by subtracting the scaled gradient $\frac{\partial \mathcal{L}}{\partial w_i}$. That is because the gradients point to the maximum direction. The positive constant η used for scaling the gradient and is called the learning rate. It determines the step of the GD search. When batches of data are used instead of the whole training dataset to compute the gradient and update the weights, the prefix "stochastic" is added to GD. The advantage of SGD is that it runs faster at every iteration, and the batch is more comfortable to fit into the memory. Although a single iteration is more rapid, SGD takes much more time to converge. To solve this issue, SGD is usually used with momentum

$$v := \beta v - \eta \cdot \frac{\partial \mathcal{L}}{\partial w_i} \tag{2.24}$$

$$w_i := w_i + v \tag{2.25}$$

Where β is momentum, and $\boldsymbol{v}$ is the velocity. Usually, $\boldsymbol{v}$ is initialized to zero and β to 0.9 for damping purposes. Interested readers are referred to chapter 8 of the book Parallel Distributed Processing [29] for in-depth review.

2.3.5.2 RMSProp Optimizer

RMSprop is an unpublished, adaptive learning rate method proposed by Geoff Hinton in Lecture 6e of his Coursera Class [30]. It divides the gradient by a running average of its recent magnitude, as in equation (2.27).

$$g \coloneqq \gamma \cdot g - (1 - \gamma) \cdot \left[\frac{\partial \mathcal{L}}{\partial w_i}\right]^2 \tag{2.26}$$

$$\mathrm{w_i} \coloneqq \mathrm{w_i} - \frac{\eta}{\sqrt{g} + \epsilon} \cdot \left[\frac{\partial \mathcal{L}}{\partial w_i}\right] \tag{2.27}$$

Where γ is a decay factor, and ϵ is a small constant that prevents diving by zero and is usually set to 10^{-8}. Hinton suggests γ be set to 0.9, while a reasonable default value for the learning rate η is 0.001.

2.3.5.3 Adaptive Moment Estimation (Adam) Optimizer

Adam is an optimization method that computes the adaptive learning rate for each parameter [31]. It can be looked at as a combination of RMSprop and Stochastic Gradient Descent with momentum. It uses the squared gradients to scale the learning rate like RMSprop, and it takes advantage of momentum by using the moving average of the gradient instead of the gradient itself like SGD with momentum.

$$m \coloneqq \beta_1 m + (1 - \beta_1) \cdot \left[\frac{\partial \mathcal{L}}{\partial w_i}\right] \tag{2.28}$$

$$g \coloneqq \beta_2 g + (1 - \beta_2) \cdot \left[\frac{\partial \mathcal{L}}{\partial w_i}\right]^2 \tag{2.29}$$

$$w_i \coloneqq w_i - \frac{\eta}{\sqrt{g} + \epsilon} \cdot m \tag{2.30}$$

m and g represent the estimates of the 1st moment (the mean) and the 2nd moment (the uncentered variance) of the gradients, respectively. In practice, they are

initialized at zero. β_1 and β_2 are decaying factors and preferable initialized to 0.9 and 0.999, respectively.

2.4 Convolutional Neural Network

Among different types of deep neural networks, Convolutional Neural Networks (CNN) has been studied extensively. CNN is ANN with at least one convolutional layer. In 1990, LeCun et al. [32] established the Bedrock of CNN framework. They developed a multi-layer artificial neural network called LeNet-5, which could classify hand-written digits. It can obtain robust features of the original image directly from raw pixels. However, The significant shift of attention towards CNN started in 2012 when Krizhevsky et al. [5] published their architecture AlexNet (as mentioned earlier in the introduction), which is similar to LeNet-5 but deeper, benefitting from the power of GPUs. In this section, we will review the essential CNN layers, namely convolutional, pooling, and fully-connected layers.

2.4.1 Convolutional Layer

The convolutional layer aims to learn feature representations of the inputs. The convolution layer is composed of several convolution kernels that are used to compute different feature maps. Feature maps can be obtained by first convolving the input with a learned kernel and then applying an element-wise non-linear activation function on the obtained results. In Figure 2.8, the kernel slides across the input, and at each location, the product of the kernel with the area it superimposes (receptive field) is taken to obtain the activation map on the right, which is then passed to the activation function. This procedure can be repeated with different kernels to collect as many feature maps as desired. The aim is to update the kernel weights to enhance the final (classification/regression) decision outputted by the whole network.

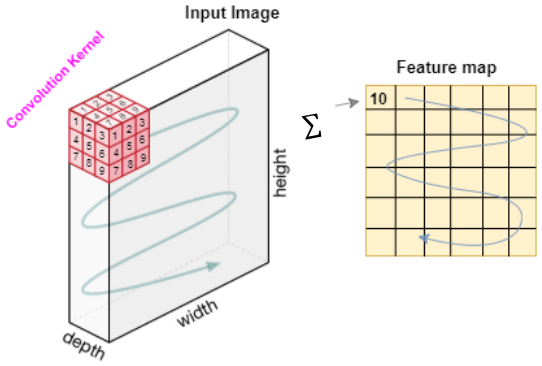


Figure 2.8. Movements path of the convolution kernel operating on the input image to output feature map [32].

Note that the number of output feature maps is equal to the number of kernels. The shape of the resultant feature map can be determined by the equation (2.31):

$$f_n = \frac{i_n - k_n + 2p_n}{s_n} + 1, \quad n = 1,2, \dots, N \tag{2.31}$$

Where f_n is the feature map shape in the axis n. i and k are input shape and kernel shape, in the axis n, respectively. p is the amount zero-padding to add on the axis n, and The amount by which the filter shifts is the stride length s.

2.4.2 Pooling layer

A common problem of feature maps is that they are sensitive to the location of the inputs. The pooling layer aims to achieve shift-invariance by providing an approach to downsampling feature maps. It is usually placed between two convolutional layers. Figure 2.9 shows two standard pooling methods; average pooling and max pooling. Note that unlike convolution kernels, the pooling layer has no learning parameters, and equation (2.31) can also be used here to determine output shape.

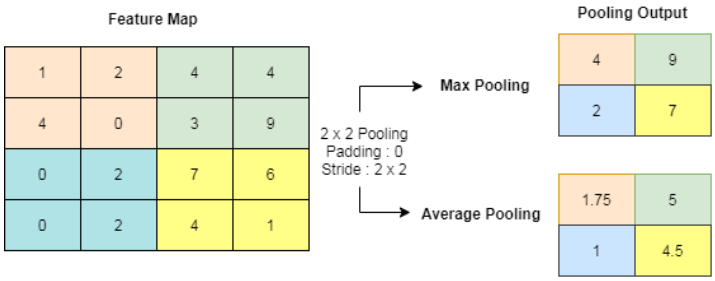


Figure 2.9. Average and max-pooling example. Better viewed with colors.

2.4.3 Fully-Connected Layers

Fully-connected (FC) layers aim to perform high-level reasoning. FC layers are put at the end of CNN architecture, i.e., after several convolution layers and max/average pooling layers. With the high-level features extracted from previous layers, FC will attempt to output a class-score or predict object coordinates from activations. Figure 2.10 shows the first CNN framework published by LeCun et al. [32]. It is common practice to use softmax activation with FC outputs. FC layer takes all neurons in the previous layer and connects them to every single neuron of the current layer to generate global semantic information. Note that a 1×1 convolution layer can replace FC layers.

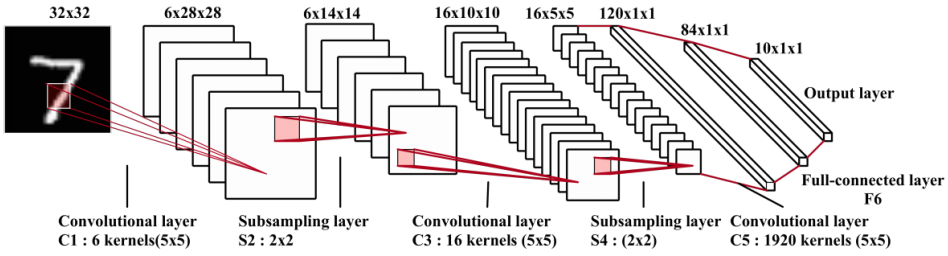


Figure 2.10. LeNet-5 network [32].

Nowadays, there are many techniques introduced to enhance the CNN learning process such as Dropouts, Regularization, ℓ_p norm Regularization, Data Augmentation, and batch normalization. The interested readers can refer to [33] for a detailed review.

2.5 Recurrent Neural Networks

In different DL architectures such as CNN and Siamese network, all inputs and outputs are independent of each other. However, natural language processing, tracking, forecasting, and many other tasks require dealing with sequential data. This means a dependency should be formed between the outputs and inputs for better predictions. Recurrent Neural Networks (RNNs) are models that can capture the dynamics of sequences via forming a state that can store the information about what has been calculated so far. The research on RNNs began in 1982 by J. Hopfield [34]. He proposed a network that is useful for recovering a stored pattern from a corrupted version. However, it didn't pay much attention until the past several years, where storage has become more affordable, sequential datasets are made available, the field of parallel computing and GPU capabilities has advanced considerably.

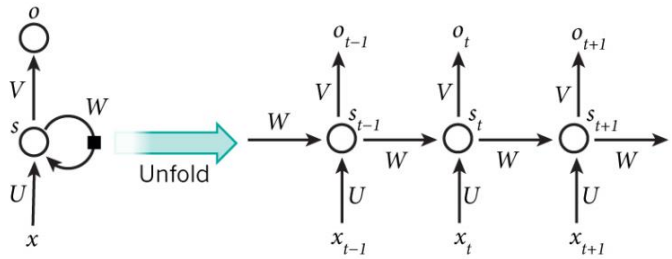


Figure 2.11. Sample RNN architecture on the left and the unfolded time representation. i.e., write the network for the full sequence [35].

x_t in Figure 2.11 represents the input at time step t. To make it clear, If this is a tracking task, x_t will represent the detection coordinates of a tracklet, and t represents the frame number. s_t is the hidden state (memory of the network); note that we might have one shared state for all time steps. U, V and W are the parameters to be learned. U, V and W weights the input, output, and the propagated hidden state, respectively. Hidden states and outputs are computed as follows:

$$s_t = \mathcal{F}(U \cdot x_t + W \cdot s_{t-1}) \tag{5.1}$$

$$o_t = softmax(V \cdot s_t) \tag{5.2}$$

The function $\mathcal{F}$ represent non-linear activation function. Usually, $tanh$ is used. Backpropagation Through Time (BPTT) is utilized to update the learnable parameters. Interested readers may refer to [36] for step-by-step derivation.

The main problem of RNN is that they don't have long memories. The reason is that when we update W in the unrolled temporal loop, the gradient descent needs to backpropagate the gradient from every time step and multiply by the state until reaching the first time step. Since the weights of neural networks are initialized close to zero, the more time steps we have, the faster W weights will vanish due to consecutive multiplications. On the other hand, if the weights are initialized above 1, W values will explode when backpropagating the gradient for the same reason. Usually, this issue is referred to as "exploding gradients."

2.5.1 Long-Short Term Memory

The beforementioned issues are the primary motivation for designing the Long-Short Term Memory (LSTM) model. LSTMs are a particular type of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997) [37]. A standard LSTM cell (Figure 2.12) has four elements:

- **Input gate:** It updates the cell state C_{t-1} . It takes the input x_t at time step t with the previous hidden state h_{t-1} to decide on which weights need to be updated for a better prediction.
- **Cell state connection:** a link that connects all gates together to output the new cell state C_t.
- **Forget gate:** Prevent the internal state weights to grow without limit (solves gradient expansion problem). It provides LSTM the ability to reset the cell state C_{t-1} as long as it needs. In other words, the essential state values are weighted by a number close to one. Otherwise, a number close to zero.
- **Output gate:** It updates the previous hidden state h_{t-1} by weighing it with the new cell state C_t.

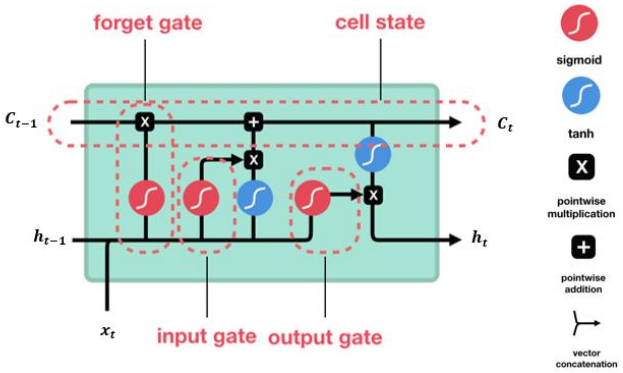


Figure 2.12. Standard LSTM cell [38].

CHAPTER 3

MOT: METRICS AND DATASETS

This chapter aims to introduce the dataset used for the MOT task and explain the evaluation protocols. In section 3.1, we present the most vital datasets, MOT, and KITTI datasets. Although they target different classes (pedestrians and vehicles tracking respectively), they are both characterized by crowd sequences, different viewpoints, various illumination levels, and camera motions. Following this, in section 3.2, we will cover the conventional metrics that are used to rank trackers.

3.1 Benchmark Datasets

In the past few years, several datasets for MOT have been published. In this section, we are going to describe the most important ones, starting from the MOTChallenge benchmark, then KITTI, and other less commonly used MOT datasets.

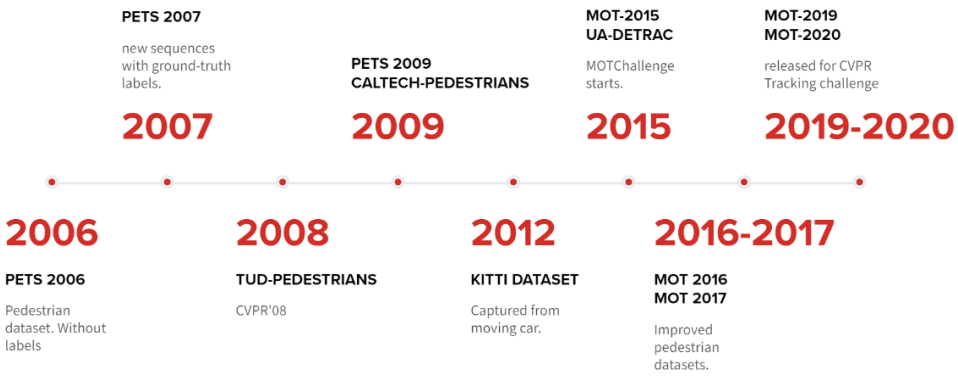


Figure 3.1. Timeline for the most used MOT datasets.

3.1.1 MOTChallenge

MOTChallenge is the most commonly used benchmark for multiple object tracking. It provides, among others, the largest datasets for pedestrian tracking. The MOTChallenged was launched with the goal of establishing a standardized evaluation of multiple object tracking methods. They chose to focus on multiple people tracking, "since pedestrians are well studied in the tracking community, and precise tracking and detection has high practical relevance" [8]. Before 2018, the submissions to MOTChallenge are made through their website at any time of the year. However, to keep track of the state-of-the-art methods from major conferences and journal, they are now made through annual organized workshops.

3.1.1.1 MOT15

The first MOTChallenge dataset is 2D MOT15 [39]. The challenge of MOT15 is not to perform well on an individual sequence, but rather to perform well on a diverse set of sequences. For this reason, a variety of sequences were filmed from different viewpoints, with different lighting conditions, and different levels of crowd density, along with including some sequences from both PETS and KITTI datasets. Sequences are very different from each other and can be classified according to camera motion (static/moving), viewpoint, and weather. For fair evaluation, both training and test sequences are picked to form an equal distribution as in Figure 3.2.

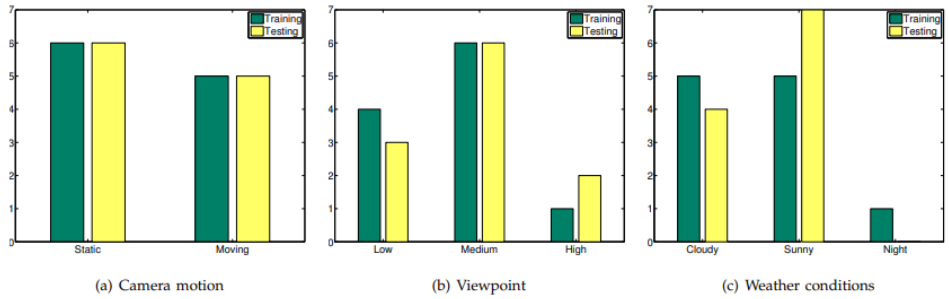


Figure 3.2. Histogram comparing train and test sequences distribution [39].

Every participant is provided three elements, train data, test data, and detections.

1) Train data: 11 sequences with a total length of 6:30 minutes and 39905 annotations.
2) Test data: it consists of 11 sequences (3 new sequences and five from other datasets) with over 10 minutes of footage and 61440 annotated bounding boxes (not available online). Although it is from the same distribution of train data, it Is hard to overfit as
3) Detections: trackers may use the available detections provided by the MOTChallenge team. Detections are collected from the aggregated channel features (ACF) model [40] trained on the INRIA dataset [41], rescaled with a factor of 0.6 to enable the detection of smaller pedestrians.

3.1.1.2 MOT16 & MOT17

A new version of the dataset was presented in 2016, called MOT16 [7]. Unlike the initial release, all videos of MOT16 have been carefully annotated following a consistent protocol. That is, not only new bounding boxes are provided but also the visibility of each one (1.0 for totally visible and 0.0 for fully occluded pedestrians). The dataset consists of a total of 14 videos (7 for training and 7 for testing), and the mean crowd density is three times higher than MOT15, which makes MOT16 more challenging for the tracking community. After the MOTChallenge team tested several state-of-the-art object detection models at that time, they found that the Deformable Part-based Model (DPM) v5 [42] outperforms ACF. Hence, they provided 215,166 detections bounding boxes. After one year, they released MOT17, which includes the same videos as MOT16, but with more accurate ground truth and with three sets of detections for each video: one from Faster R-CNN [28], one from DPM, and one from the Scale-Dependent Pooling detector (SDP) [43]. The detections collected from the three beforementioned models are called public detections. Similar to MOT15 Figure 3.2, MOT16 & MOT17 test sequences were chosen from the same distribution as train sequences, as shown in Figure 3.2.

Figure 3.3. : An overview of the MOT16 & MOT17 datasets. Top: Training sequences; bottom: test sequences [7].

3.1.1.3 MOT19 & MOT20

Due to the tremendous contribution the first MOT releases have to the community, a dedicated MOT Challenge workshop was established at the Computer Vision and Pattern Recognition Conference (CVPR) 2019 & 2020. Both MOT20 and MOT19 were present in that workshop, and they consist of 8 new sequences (4 testings and four training sequences) depicting very crowded, challenging scenes, as shown in Figure 3.4. The average density per frame is 246 pedestrians [8].

Figure 3.4. Example frames from MOT20 sequences [8]. (a) Sequence 05, (b) Sequence 08, (c) Sequence 03 , (d) Sequence 01.

3.1.2 KITTI Vision Benchmark

While MOTChallenge datasets are dedicated to pedestrian tracking, the KITTI benchmark focuses on pedestrians and vehicles [44] for the task of autonomous

driving. The benchmark was established in 2012 and is managed by the vision lab of Karlsruhe Institute of Technology (KIT). The dataset was captured by driving a car around Karlsruhe city in Germany (Figure 3.5). It consists of 21 training and 29 test videos, with a total of about 19000 frames (32 minutes). Similar to MOT, they provide detections collected by the DPM model.

Trackers can choose to be evaluated on pedestrian labels, vehicles, or both, and according to their option, they will be placed in the pedestrian leaderboard, vehicles one, or both, respectively. Note that the benchmark uses the same metrics for evaluation as the MOTChallenge; this is why it is common to find similar entries in the KITTI pedestrian leaderboard and the MOTChallenge one.

Along with detections and ground-truth labels, KITTI provides the difficulty class of each bounding box in the following manner:

1. **Easy:** Min. bounding box height: 40 Px. Occlusion level: Fully visible, Truncation: 15 %
2. **Moderate:** Min. bounding box height: 25 Px. Occlusion level: Partly occluded, Truncation: 30 %
3. **Hard:** Min. bounding box height: 25 Px. Occlusion level: Difficult to see, Truncation: 50 %.

Figure 3.5. The car is used to capture KITTI sequences. It provides raw 2D stereo camera data, 3D GPS, and Velodyne Laserscanner data [45].

3.1.3 Other Benchmarks

Besides the previous datasets, there are a couple of other, old, and less frequently used datasets.

- **UA-DETRAC Benchmark:** University at Albany DEtection and TRACking (UA-DETRAC) dataset were released in 2015 [46]. It consists of 100 challenging videos, 10 hours of videos (140,000 frames) captured from static surveillance cameras around China. It provides rich annotations, including illumination, vehicle type, occlusion, truncation ratio, and vehicle bounding boxes, as shown in Figure 3.6.

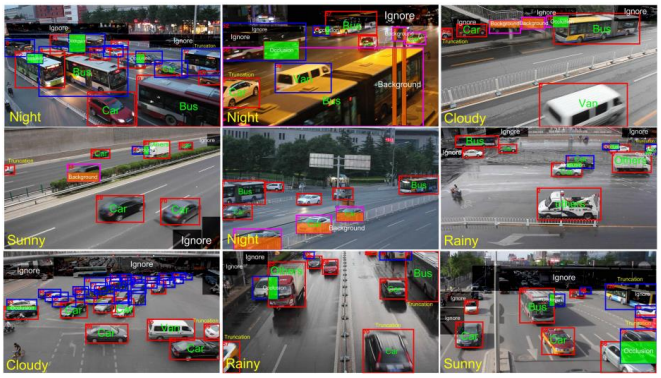


Figure 3.6. Sample annotated frames [46].

- **TUD Benchmark:** TU Darmstadt pedestrian tracking dataset [47] was captured from the TU Darmstadt campus to evaluate M. Andriluka et al. proposed tracker. After publishing their work in CVPR' 08, the dataset was made publicly available. It consists of 610 train images and 250 for testing. For each training image, annotations of body-part positions, as well as detailed segmentation of people, are provided (Figure 3.7). Their work showed that tracking body parts (instead of the single bounding box) improve hypotheses for the position of each person in several subsequent frames.

Figure 3.7. Sample train image with annotations [47].

- **PETS:** up to our knowledge, PETS was the first publicly available MOT dataset. It was established in 2006 but significantly improved in 2009 for the IEEE Workshop on Performance Evaluation of Tracking and Surveillance [48]. The goal of PETS is to track individuals within crowds. The dataset was captured with eight static cameras from 8 different views. It is divided into four subsections (Figure 3.8); S0 is the training set, S1 concerns person count and density estimation, S2 addresses people tracking, and dataset S3 involves flow analysis and event recognition.

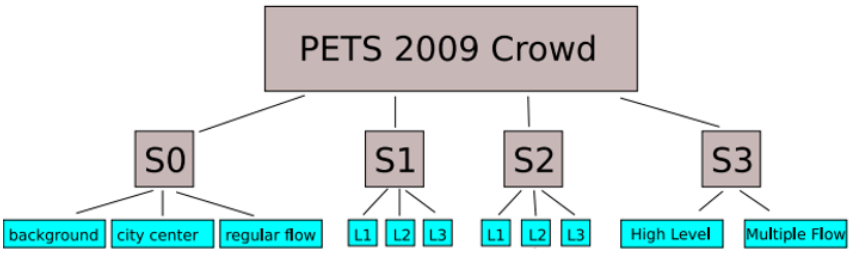


Figure 3.8. Hierarchy of the PETS2009 sequences and related views [48]. L1-L3 refers to the difficulty of the challenge.

3.2 Evaluation Metrics

In order to provide a fair experimental setup where algorithms can be tested and compared, a group of metrics has been established, and they are used in almost every work. Note that the evaluation metrics used for object detection models can't be used directly here. Detection task metrics focus on the precision of the bounding box, how confident are the detections, and how many objects it miss or wrong detections. For this reason, several metrics are defined for the MOT task. In this chapter, we will begin with the classical metrics. We then present CLEAR MOT & ID metrics.

3.2.1 Mean Average Precision

The standard way to measure the quality of bounding boxes includes calculating the mean average precision (mAP) at given Intersection Over Union (IoU) thresholds. The next two subsections illustrate how IoU is computed and utilized for mAP calculation.

3.2.1.1 Intersection over Union (IoU) :

Intersection over Union is simply an evaluation metric to compare the ground-truth bounding boxes with the predicted ones. IoU can be computed between two bounding boxes b_1 and b_1 by the pseudocode provided below. IoU is better explained in Figure 3.9. In case (a) IoU is poor for two cases, when the predicted object does not overlap properly with the ground-truth one, or when it overlaps (totally) but with other regions from the background. IoU is excellent if and only if it fits the ground-truth object precisely as in (c).

Algorithm 1: Computing Intersection Over Union (IoU)

Input $\boldsymbol{b_1}$ & $\boldsymbol{b_2}$ matching the format:

$$(\boldsymbol{x_{(top-left)}}, \boldsymbol{y_{(top-left)}}, \boldsymbol{x_{(bottom-right)}}, \boldsymbol{y_{(bottom-right)}})$$

Output The Interesection Over union between $\boldsymbol{b_1}$ & $\boldsymbol{b_2}$.

Procedure $\boldsymbol{IoU \leftarrow IoU(b_1, b_2)}$

Find the Area of intersection

$$\boldsymbol{x_A = max(b_1[0], b_2[0])}$$

$$\boldsymbol{y_A = max(b_1[1], b_2[1])}$$

$$\boldsymbol{x_B = max(b_1[2], b_2[2])}$$

$$\boldsymbol{y_B = max(b_1[3], b_2[3])}$$

Inter-Area $\boldsymbol{\leftarrow max(0, x_B - x_A + 1) \times max(0, y_B - y_A + 1)}$

Find the Area of each bounding box

$$\boldsymbol{Area_A = (b_1[2] - b_1[0] + 1) \times (b_1[3] - b_1[1] + 1)}$$

$$\boldsymbol{Area_B = (b_2[2] - b_2[0] + 1) \times (b_2[3] - b_2[1] + 1)}$$

$$\boldsymbol{IoU = \frac{Inter-Area}{(Area_A + Area_B - Inter-Area)}}$$

End procedure

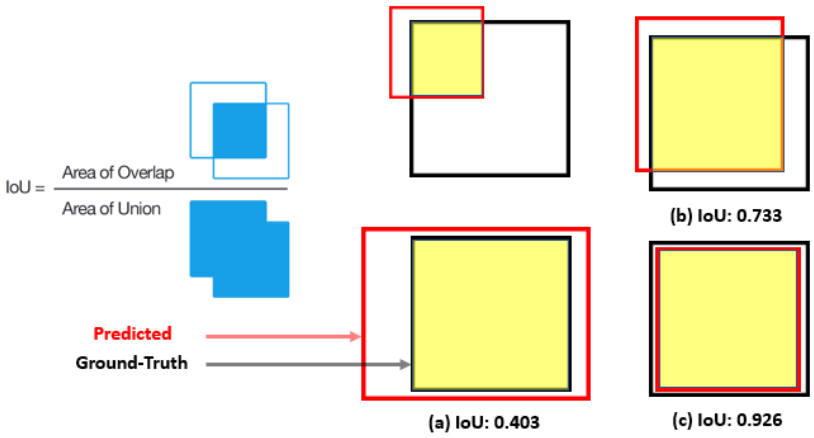


Figure 3.9. An example of computing IoU for various bounding boxes.

3.2.1.2 Average Precision (AP) Computation

Average Precision for a single class is computed by feeding the ground-truth and the predicted bounding boxes to the pseudocode below. Note that the AP metric does not take False Negatives into account (e.x. detecting non-real object), which is one of its disadvantages. The **estimated area** in the pseudocode refers to one of the conventional methods specified by the dataset benchmark, such as 11-points interpolation (Pascal VOC 2008 competition [49]), 101- points interpolation (COCO dataset [50]), or Area Under Curve (Pascal VOC 2010-2012 competitions [51]–[53]). Different datasets require different AP criteria. For example, In the COCO dataset, AP is provided with a subscript for the IoU threshold used to filter predictions or the scale of the bounding box.

```
Average Precision (AP):
   AP                  % AP at IoU=.50:.05:.95 (primary challenge metric)
   AP^IoU=.50          % AP at IoU=.50 (PASCAL VOC metric)
   AP^IoU=.75          % AP at IoU=.75 (strict metric)
AP Across Scales:
   AP^small            % AP for small objects: area < 32^2
   AP^medium           % AP for medium objects: 32^2 < area < 96^2
   AP^large            % AP for large objects: area > 96^2
```

Figure 3.10. The six metrics used for ranking in the COCO dataset [54].

Algorithm 2: Computing The Average Precision (AP) for a single class

Input list of ground-truth $\boldsymbol{GT_i}$ and predicted $\boldsymbol{D_i}$ bounding boxes for every frame in the sequence ($\boldsymbol{i}$ represent the frame number).

Output The Average Precision AP (AP) between

Procedure $\boldsymbol{AP \leftarrow AP(GT, D)}$

for $\boldsymbol{f}$ in length($\boldsymbol{D}$):

for $\boldsymbol{d}$ in $\boldsymbol{D_f}$:

$$\boldsymbol{d(conf_{score}) = max\left(IoU\left(GT_f, d\right)\right)}$$

if $\boldsymbol{d(conf_{score}) \geq conf_{threshold}}$:

$$\boldsymbol{TP_{(list)} \leftarrow 1}$$

$$\boldsymbol{FP_{(list)} \leftarrow 0}$$

else:

$$\boldsymbol{TP_{(list)}} \leftarrow \mathbf{0}$$

$$\boldsymbol{FP_{(list)}} \leftarrow \mathbf{1}$$

$\boldsymbol{TP_s}$ & $\boldsymbol{FP_s} \leftarrow$ sort $\boldsymbol{TP_{(list)}}$ & $\boldsymbol{FP_{(list)}}$ in descending order according to the confidence scores.

$$\boldsymbol{Precision} \leftarrow \frac{\sum \boldsymbol{TP_s}}{\sum \boldsymbol{TP_s} + \sum \boldsymbol{FP_s}}$$

$$\boldsymbol{Recall} \leftarrow \frac{\sum \boldsymbol{TP_s}}{\sum_f \boldsymbol{GT_f}}$$

AP ← form $\boldsymbol{Recall - Precision}$ plot and compute the **estimated area[1].**

End procedure

1: Check section 3.2.1, point **Error! Reference source not found.**.

3.2.1.3 Mean Average Precision (mAP)

Mean average precision is computed by merely averaging AP (calculated by algorithm 2) for each class in the dataset.

3.2.2 The Classical MOT Metrics

The classical metrics, defined by Wu and Nevatia [55], highlight the different types of errors a MOT algorithm can make. These metrics are computed from the tracklets formed by the tracker algorithm. A tracklet is a set of bounding boxes corresponding to one object throwout the video. That is, if a_i^t represent the bounding box for object i at time t then $A_i = \{a_i^1, a_i^2, .., a_i^n\}$ is its tracklet where n is the total length of the sequence.

Figure 3.11 visualize the following metrics :

- **Mostly Tracked (MT) trajectories:** number of ground-truth (GT) tracklets that are tracked correctly in at least 80% of the frames.
- **Fragments:** trajectory hypotheses which cover at most 80% of a ground truth trajectory. Note that more than one fragment can cover one GT.
- **Mostly Lost (ML) trajectories:** number of GT tracklets that are correctly tracked in less than 20% of the frames.
- **False trajectories:** predicted tracklets that couldn't be matched with GT.
- **ID switches:** number of times an object is tracked correctly, but its ID is changed due to occlusion or other reasons.

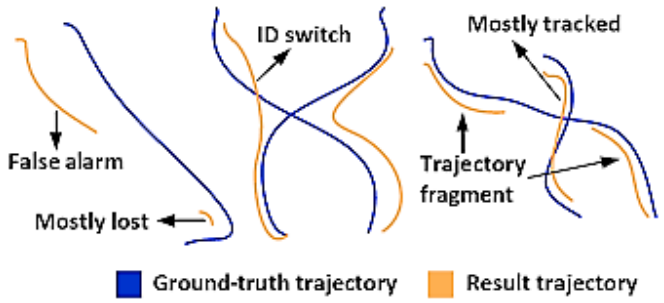


Figure 3.11. Classical tracking evaluation criteria [55]. The blue trajectories indicate ground-truth tracklet, while the golden ones are predicted.

3.2.3 CLEAR MOT Metrics

As made obvious in the chapter introduction, different datasets use different metrics to compare multiple object tracking models. However, in 2008 Bernardin and Stiefelhagen [56] proposed two standardized metrics to be used for multiple object tracking, referred to as the CLEAR MOT metrics, which include both MOT Accuracy (MOTA) and MOT Precision (MOTP). First, a mapping between ground truth and predictions is established. That is if the ground truth object $a_i^{(t-1)}$ and the prediction $b_j^{(t-1)}$ are matched in frame $t-1$, and in frame t, the $IoU\left(a_i^{(t)}, b_j^{(t)}\right) \geq$

0.5, then $a_i^{(t)}$ and $b_i^{(t)}$ are matched in that frame, even if there exists another prediction $b_k^{(t)}$ such that $IoU\left(a_i^{(t)}, b_k^{(t)}\right) > IoU\left(a_i^{(t)}, b_j^{(t)}\right)$. The ground-truth bounding boxes that cannot be associated with a prediction are counted as false negatives (FN), and the hypotheses that cannot be associated with a real bounding box are marked as false positives (FP). The scores are then defined as follows:

3.2.3.1 The MOTA

$$MOTA = 1 - \frac{\sum_{j,i}\left(FN^j{}_i + FP^j{}_i + IDSW^j{}_i\right)}{GT} \in (-\infty, 1] \tag{3.1}$$

Where i represent frame in the sequence j. GT is the total number of GT boxes. It is important to note that MOTA can be negative because the tracker can commit more errors than the number of GT boxes.

3.2.3.2 The MOTP

MOTP is computed by equation (3.2). Note that MOTP does not take into account any tracking performance and rather focuses on the quality of the detections. d_t^i is the distance between the GT and prediction of object i, and c_t is the number of matches found in frame t.

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{3.2}$$

3.2.4 Identification (ID) Metrics

One might notice that the MOTA metric focuses on FP, FN, and IDSW. i.e., the number of times the trackers make mistakes. However, we might be interested in trackers which can track the object for a longer time. For this reason, E. Ristani et al. [57] proposed new metrics that are supposed to complement CLEAR MOT metrics. In contrast to the MOTA, which matches the GT with the predictions frame-by-frame, the proposed ID metrics perform matching globally. In order to solve this issue, a bipartite graph is formed, and a minimum cost solution is taken as the problem solution. The bipartite graph (Figure 3.12) consists of two sets, V_t which include a node for each GT tracklet $\{v_1, v_2, ...\}$ and $\mathcal{F}_{c_i}^{+}$ false positive node for each computed tracklet. The second set is V_c, include a node for each predicted tracklet $\{c_1, c_2, ...\}$ and $\mathcal{F}_{v_i}^{-}$ false negative for each GT one. With this stepup , four possible pairs are present. The edge cost are set as a binaries, that is for counting purpose.

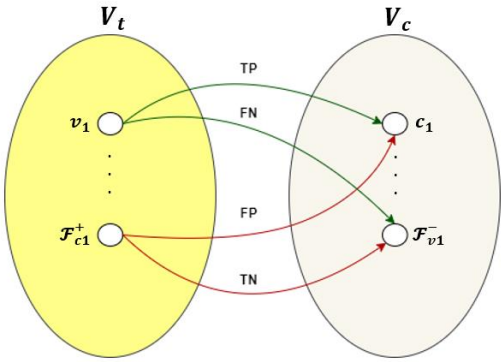


Figure 3.12. Example ID Bipartite graph with one tracklet for demonstration.

Given all the details above, the following ID scores are computed:

- IDTP: Sum of the edges selected as TP matches.
- IDFN: Sum of the weights from the selected $\mathcal{F}_{v_i}^{-}$ edges.
- IDFP: Sum of the weights from the selected $\mathcal{F}_{c_i}^{+}$ edges.

Another three important measures. Namely, Identification Precision (IDP), Identirciation Recall (IDR), and Identircaiton F1 (IDF1) are computed by using the previous scores as follows:

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{3.3}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{3.4}$$

$$IDF1 = \frac{2}{\frac{1}{IDP} + \frac{1}{IDR}} \tag{3.5}$$

The metrics that are used in almost every piece of work are the CLEAR MOT metrics, MT, ML, and IDF1. In MOTChallenge leaderboards, the number of frames per second (FPS) is also included in addition to the metrics mentioned above.

CHAPTER 4

APPEARANCE HANDLING

Appearance handling is the stage where the tracker collects information about objects in the scene. In the machine vision field, this information is referred to as features. The aim of utilizing DL at this stage is to gather a low number of features, which include high-level representation of the object. In DL-based trackers, this is done in two steps, first detecting objects in the scene by using pre-trained models and then extract essential features by passing it to the feature extractor. In this chapter, we will discuss the commonly used object detectors and various DL based feature extractors.

4.1 Object Detection

With the rapid development of deep learning networks for detection tasks, the performance of object detectors has significantly been improved. The first step of any MOT task is to localize the objects in the scene. Object detection models can be categorized into two groups, namely single-shot detectors and two-stage detectors. The former, also referred to as regression/classification based detectors, make a fixed number of predictions on a grid such as SSD and YoloV3. The latter, referred to as region proposal based detectors, leverages a proposal network to find objects and then use a second network to fine-tune these proposals and output a final prediction such as the Faster R-CNN network.

4.1.1 YOLOv3

The YOLO (You Only Look Once) algorithm proposed by Redmon et al. [58] classifies and locates the object in only one step and obtains the position and category of the object directly at the output layer. Among the recent work that used YOLO for their tracker is [59], there it is used to detect the small marine organism. Kim et al. [60] also employed YOLO to detect pedestrians from a moving vehicle camera. The early versions of YOLO lack some of the essential elements that state-of-the-art detectors have, such as residual blocks, skip connections, and upsampling. YOLOv3 added the beforementioned elements along with three scale object detection (9 anchors, three for each scale), as shown in Figure 4.1. Doubling the number of convolution layers from 53 to 106 significantly improved the performance compared to YOLOv2 but with the cost extra execution time. Yet, YOLO still performs better than SSD and Faster R-CNN when speed is given preference over accuracy [61].

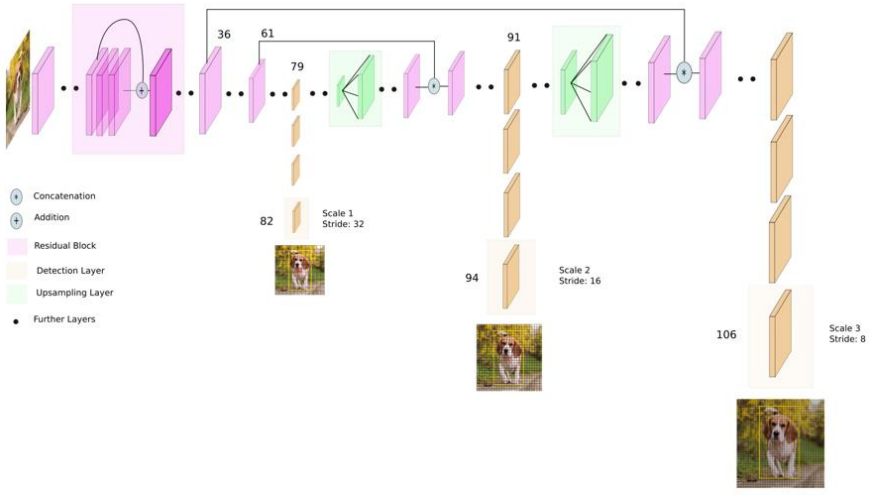


Figure 4.1. YOLOv3 architecture.

At first, YOLOv3 takes an image with the shape 416×416. The image is then forwarded to 81 convolutional layers. The first detection (for the first scale) is done at the layer 82^{nd}. At this point, the feature map is downsampled with strides of 32, resulting in a new feature map of size 13×13. Moreover, the depth of this feature

map is determined by $1 \times 1 \times (B(5 + C))$ Where B represents the number of bounding boxes, a cell on the feature map can predict, C is the number of classes, and 5 represents the attributes of the bounding box (Figure 4.2). This relation is better viewed in the figure below. At the same time, the 79th feature map is then subjected to upsampling then another convolutional process for the next detection stage (for the 2nd scale) at layer 94.

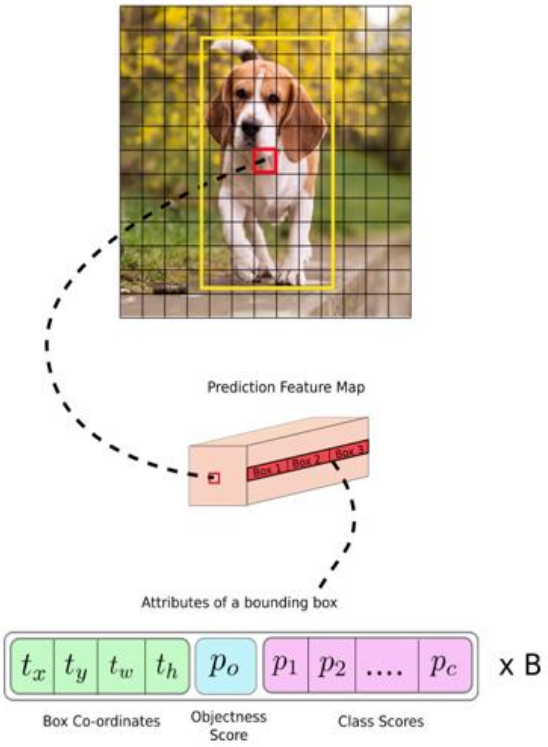


Figure 4.2. Attributes of the bounding boxes of YOLOv3.

4.1.2 Single-Shot Detector

Another commonly used detector is the Single-Shot Detector (SSD) [62], which was released at the end of November 2016. It reached new records in terms of performance and precision for object detection tasks, scoring over 74% mAP (mean Average Precision) at 59 frames per second on standard datasets such as COCO [63]. Recent publications have shown that SSD performs better in tracking compared

to YOLO. R. Deepa [64] et al. concluded in their work that among the three detection models (from this chapter), SSD is a much efficient and more accurate algorithm in tracking tennis ball in the real-time stream. P. Nousi. et al. [65] reached the same conclusion for tracking objects from Unmanned Aerial Vehicles (UAV). In fact, they were disappointed to have a massive number of false positives predicted by YOLO. In addition to that, SSD can be trained to take features as an input instead of the whole frame (pixels domain) to make predictions. In [66], the frame is first fed to a custom CNN network along with the trackers' previous coordinates. Then, multiple feature maps are generated and fed to SSD to output the final predictions. Their algorithm showed state-of-the-art performance on MOT15 and KITTI datasets. To better understand SSD, we will review its architecture depicted in Figure 4.3. SSD is run once on an input image with a size 300×300. Similar to YOLO [58], the image is divided into units, and every unit will include a predefined number of bounding boxes. At first, the backend network (VGG) generates a set of feature maps, which are then passed to the consecutive convolutional layers. The aim of having multiple consecutive convolutions is to downsample the heatmap and collect complex features to enhance small object detection. Between every two convolutional layers, the feature maps are examined by a Detector and Classifier (D&C) subnetwork (such as the highlighted box in yellow Figure 4.3). D&C is designed to perform a scan for a specific scale. Figure 4.4 depicts an example D&C network where two convolution kernels are present. The former is trained to output top-left coordinate (x, y), wdith and height predictions for every generated bounding box while the latter compute the confidencese scores. Finally, all estimated bounding boxes and scores are concatentaed and only the ones with a high confidence score (above a certain threshold, usually 0.5) are kept. It is a common practice to run non-maximum suppression on the resultant bounding boxes to eliminate the overlapping ones.

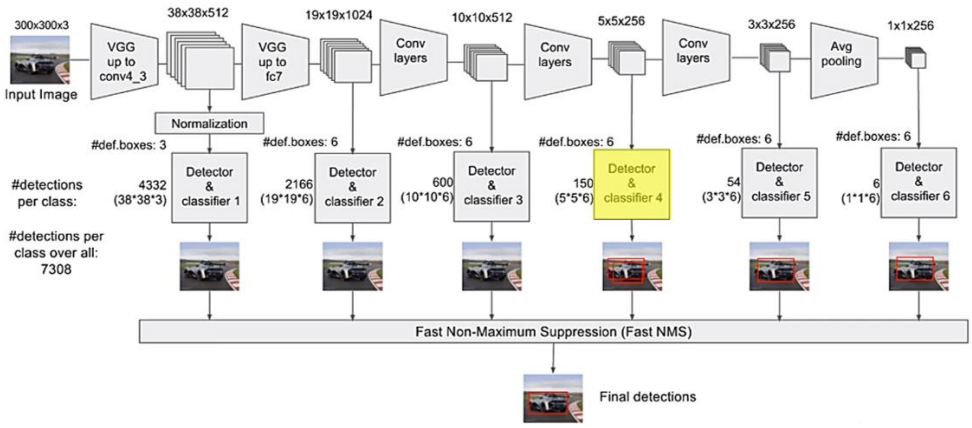


Figure 4.3. SSD network architecture [67].

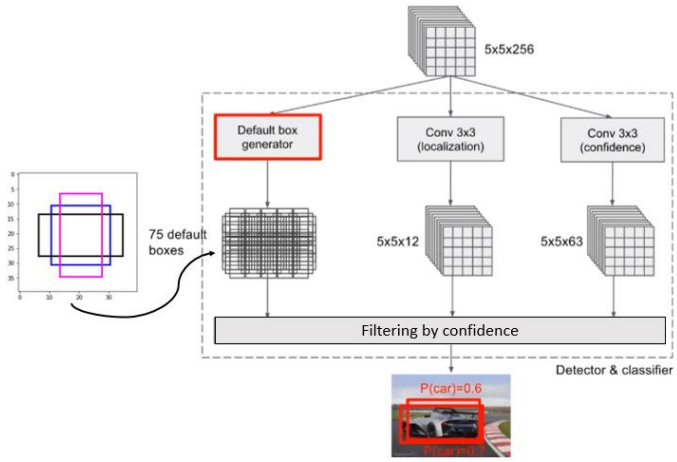


Figure 4.4. Fourth D&C network in the SSD network [67].

4.1.3 Faster R-CNN

When it comes to pedestrian detection, the default choice for the state-of-the-art MOT algorithms is the Faster R-CNN network [28] due to its leading accuracy on several pedestrian benchmarks [68]. The usage of Faster R-CNN in the tracking filed began in 2016 when A. Bewley et al. published the SORT algorithm [69]. SORT algorithm managed to improve the MOTA score by 18.9% [39], which was a significant improvement at that time. The paper [70] is another notable work that modified the Faster R-CNN architecture by adding skip-pooling [71] and multi-region features [72] (i.e., collecting complex features) to achieve better tracking

performance. Faster R-CNN is a multi-stage pipeline that consists of a deep, fully convolutional network that proposes regions (RPN) and a Fast R-CNN detector head that uses the proposed regions, as shown in Figure 4.5.

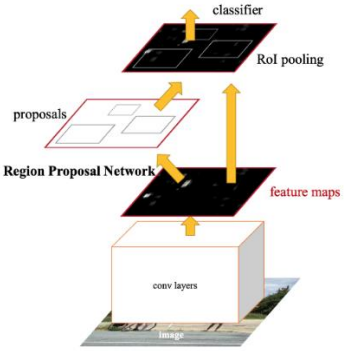


Figure 4.5. Faster R-CNN architecture overview [28].

In contrast to single-shot detectors, Faster R-CNN slides on the generated feature map instead of dividing the input image to a grid. This is one reason why it performs better on small objects compared to YOLO and SSD. At first, then input image is passed through convolutional layers (Backbone network similar to VGG16) to produce a $60 \times 40 \times 256$ feature map. Then, a 3×3 sliding kernel with 256 units is applied to the backbone feature map, as shown in Figure 4.6, to give a 256-d feature vector for every location. Finally, two 1×1 convolution kernel is used with $2k$ and $4k$ units to output classification scores and coordinates regression, respectively. The k here represent the total number of anchors (on the right side of Figure 4.6) is by default set to 9.

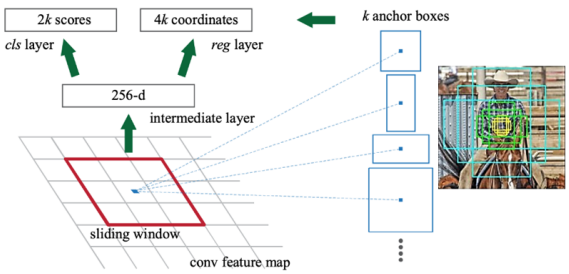


Figure 4.6. Region Proposal Network (RPN) [28].

4.1.4 Other Usage of Detection Models

Detection models can be used for different purposes than object localization. For example, in [73], Faster R-CNN was used to eliminate false positives to track multiple vehicles by first collecting detections using a classical method (background subtraction + SVM) then the CNN model decides whether to keep them or not. P. Bergmann et al. [74], in their work, trained Faster R-CNN to take the current frame with the previous bounding box coordinates to output the expected location in the consecutive frame. In other work, localization models were used to extract features (from mid-layers) instead of having a separate network.

4.2 Feature Extraction

The extraction of robust features from a complex scene while tracking is an essential step in MOT. If the object and its features are detected once reliably, the tracking is proofed to last longer. In tracking, features are collected to distinguish between objects not only in the same frame but on the consecutive ones too. In the literature, feature extraction is sometimes referred to as re-identification features that is because they are intensively used for this purpose. CNN and Recurrent NN are the preferable topologies to collect appearance information to match similar objects while the Siamese networks are used to reidentify targets in the scene.

4.2.1 CNN Based Feature Extractor

Up to our knowledge, Kim et al. [75] is the first work to apply CNN based feature extraction for the MOT task. In their work, all detections are fed to a pre-trained CNN model, which generates 4096 features for each bounding box; these features are then compressed using Principle Component Analysis (PCA). They were able to enhance the MOTA score by 3 percent, and they achieved the top tracker at that time in the MOT15 dataset.

SORT algorithm (mentioned in 4.1.3) was also improved in 2017 [76] by integrating an Appearance Information Network (AIN). AIN is a customized CNN network with 11 layers. It takes every predicted bounding box and outputs a vector with 128 features. Finally, the minimum Mahalanobis distances between all vectors are computed for the association task. According to their experiments, they were able to reduce ID switches by 45% and track objects for a longer time.

Other trackers use classification models for extracting features [77]. That is, the classification model is trained to distinguish between pedestrians and the background. Then, while tracking, the classification layer is removed, and the features from the last fully connected layer are used. We have noticed that one of the most common classifiers used in the state-of-the-art trackers is the ResNet family proposed by Microsoft [78] due to its outstanding performance in ImageNet challenge [79].

Because execution time is vital in tracking tasks, features sometimes are collected along with detections from the localization models instead of having a separate network for that. For example, the appearance features are cropped from RoI of the Faster R-CNN network (Figure 4.5) corresponding to a specific target. In other words, the backbone network of the detection model can also be used to collect robust features.

A combination of classical and CNN based features are also applicable. [80] is one of the recent work that proposes a multiple object tracker, called MF-Tracker, that combines classical features (spatial distances and colors) along with modern ones collected from the Omni-Scale Network (OSNet) [81]. Basically, the tracker generates a single similarity score for every detection to compare the targets across the frames. Their tracker reached state-of-the-art performance on the UA-Detrac dataset with MOTA 76.3%. CNN based features are also utilized to reidentify occluded objects or objects reinterring the scene [81]–[83]. However, Siamese based architectures are more commonly used for this task because they learn not just the target appearance but also the distance measurement.

4.2.2 Siamese Based Feature Extraction

This section will discuss the critical role of Siamese based feature extractors in the tracking task. However, before the discussion, we will briefly explain the commonly used Siamese topologies in MOT.

4.2.2.1 Siamese Network

Although the Siamese network recently has gone through enormous changes, we found the definition provided by R. Pflugfelder [84] suitable to summarize all of the current topologies. There it is defined as a Y-shaped neural network that joins two network branches to produce a single output. Specifically, in the scope of machine vision, it takes two image pairs (or two groups of images) to predict the similarity between them. Similarity learning with CNNs started with the pioneering work [85]. The network depicted in Figure 4.7 takes the last bounding box of a specific target z reshaped to $127 \times 127 \times 3$ and a search image (4 times the area of the target) x. Then, the similarity is computed at all translated sub-windows on x in a single evaluation by convolutional operation between the two generated feature maps (by the backbone network φ). At that time, their algorithm ranked first in the Visual Object Tracking (VOT) challenge [source].

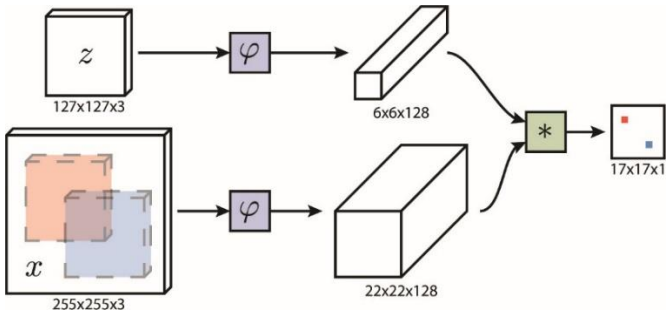


Figure 4.7. SiameseFC architrecture [85].

A large number of improvements followed the release of SiameseFC [86]–[89]. For example, on [87], the same authors modified the baseline SimaeseFC network by

integrating a correlation filter between z and its feature map. This change significantly enhances its performance on six different benchmarks [90]. You may refer to the original work for in-depth details.

Siamese topologies can be categorized into three (Figure 4.8):

- **Cost Function:** input pairs are fed to two separate backbone networks (the networks may have shared weights), and a cost function (e.x. as Euclidean distance) is applied to the outputs to measure the similarity.
- **In-Network:** instead of the cost function, a DL based network is trained to output the similarity metric by fusing backbone networks output.
- **Input Stacking:** instead of a Y shaped network, the inputs are stacked and fed to a single backbone network.

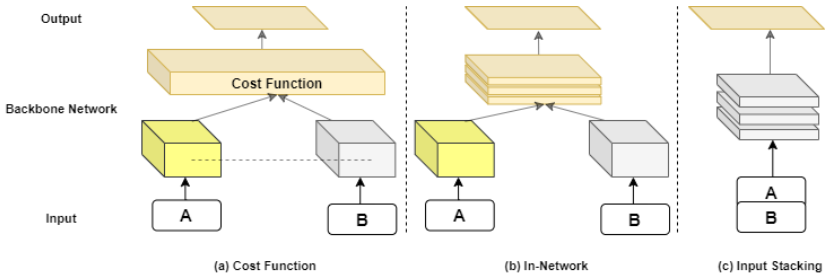


Figure 4.8. Summary of common Siamese topologies.

It is interesting to note that for the last three years, almost all top-ranked trackers in VOT are Siamese based [91]–[93]. However, the main disadvantage of the Siamese network is the execution time relative to the number of objects being tracked. i.e., if we assume that tracking a single feed-forward pass (tracking single target) requires 10 ms (100 FPS) tracking a scene of 100 objects will require 1 s (1 FPS) in a scene with one object will require. This issue is vital when it comes to MOTChallenge, specially MOT20, which has an average density of 200 pedestrians per frame.

4.2.2.2 Siamese In MOT

Siamese networks can be used in two ways in the task of MOT. Either train them and then collect features in an early stage from mid-layers or utilizing the whole network only when it is required (specific objects such as occluded or reinterring the scene targets) to minimize ID switches.

L. Leal-Taixé et al. [94] is one of the first works to use the Siamese to extract features for multiple pedestrian tracking. A two-stage tracker was proposed, at first, pairs of two detections from consecutive frames are formed. Then they collect 512 of what they call "local spatio-temporal features" from the last fully connected layer of custom Siamese network (Input Stacking topology). They managed to significantly reduce the false positive number on the MOT15 benchmark [39].

Kim et al. [95] proposed another custom Siamese model that takes two detections, their IoU, and their area ratio, as input. Similar to [94], the label layer was removed, and features are collected from the last layer. In contrast to the previous work, they focused on reducing system complexity, and the hyperparameters need to be tuned along with providing the system with more information. The network was an experiment on the MOT16 dataset [7] and could compete with batch trackers. Although it is not accurate to compare the speed of their algorithm due to lack of information on hardware specification, they mentioned that fusing geometric information to the network enhanced the performance while keeping no latency.

Another approach is to use the Siamese network when it is needed. i.e., it can be used when the tracker wants to confirm an object reinterring the scene or is not occluded anymore. [74] the current top online tracker utilizes this approach in its pipeline. The last predicted bounding box from all deactivated tracklets (occluded or leaving scene objects) are fed to the Siamese network (Cost Function topology), and the feature map is saved to memory. When new potential tracklets are present, the stored feature vectors are compared with new ones to update their status.

CHAPTER 5

GROUPING STEP

From the previous chapter, we have discussed how detections and features are collected. However, the MOT task to be completed, a trajectory needs to be assigned for every object in the scene. i.e., detections D at frame I should be associated with the correct tracklets T. The conventional way to cope with this problem is to construct an affinity matrix (cost matrix) between D & T, by using some distance measurement over the extracted features or directly to the RGB detections. Unlike SOT, forming a robust affinity matrix is quite tricky (even with precise detection) due to intra-class variations (especially in dense sequences) and other challenges discussed in section 1.1. Recent DL based tackers usually use either long-short term memory (LSTM) networks to compute affinities (end-to-end approach) and then integrate classical methods for the association. In this chapter, we will describe how state-of-the-art trackers managed to improve the traditional means by integrating DL in both steps.

5.1 LSTM Topologies in MOT

The beforementioned topology of LSTM in section 2.5 may differ from work to work, depending on the targeted task. Almost in every MOT work we reviewed (that uses LSTM), a different LSTM architecture is proposed. C. Kim et al. [96] proposed Bilinear LSTM (bLSTM) that can collect appearance information to enhance the tracklet-detection association. It is merely the conventional LSTM trained on ResNet features of a tracklet, and the detection features are used to weight the LSTM appearance output. In other words, the LSTM is forced to learn tracklet appearance only while the ResNet model (for detections, on the right Figure 5.1) is learning how to search for the best match state. The advantage of their work is that one tracklet

state can be used to compare all detections, but in the conventional LSTM, all possible pairs should be examined separately.

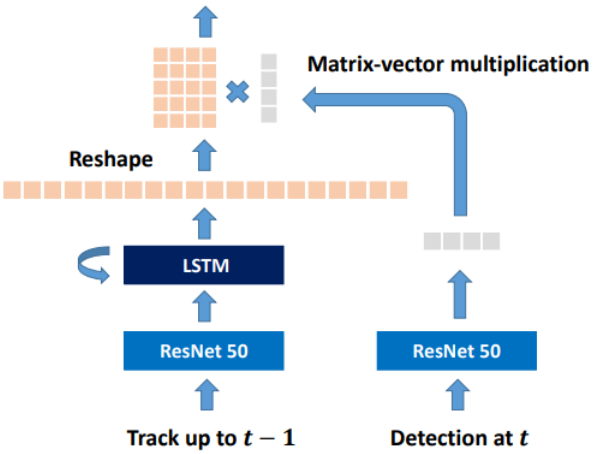


Figure 5.1. Proposed bLSTM topology [96].

L. Lan et al. [97] proposed a similar network to the previous work but with applying the FC layer instead of matrix multiplication and MaxPooling on LSTM states (Figure 5.2). With these updates, they managed to rank third in the MOT17 dataset scoring $MOTA$ 1.1% less than the top online tracker. The discussion on LSTM topologies will go along with the massive amount of publications. In this section, we pointed out two of the new LSTM topologies that were successful and had a state-of-the-art performance.

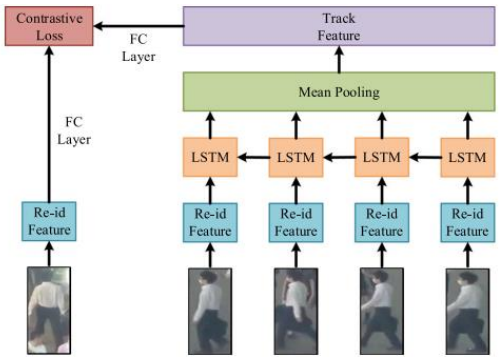


Figure 5.2. Enhanced bLSTM topology by [97]. Re-ID features are collected from the LOMO appearance model proposed by S. Liao et al. [98].

5.2 Affinity Step

In MOT, affinity scores represent either the distance measurement or the probability likelihood of two targets being similar (Figure 5.3). LSTM network is the standard way to compute DL based affinities. RNNs, on the other side, are not commonly used because of the gradient vanishing and exploding problems, which is discussed in section 2.3.1. Section 5.2.1 will be dedicated to LSTM based approaches. Yet we will discuss one of the very new and unusual algorithms in section 5.2.2, which we believe is a game-changer.

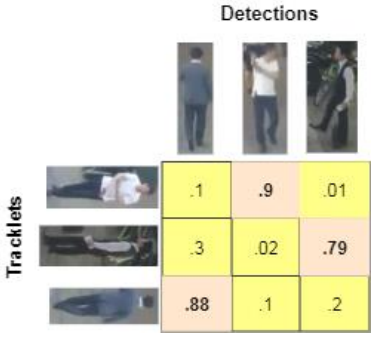


Figure 5.3. Sample of affinity matrix where entries represent probabilities or distance between features.

5.2.1 LSTM-Based Affinity Computation

One of the first trackers to utilize DL for affinity computation is [99]. The main idea of their approach, summarized in Figure 5.4, is to enhance the Euclidean distance-based affinity matrix by using temporal information (LSTM cell & hidden states). i.e., the LSTM block takes a matrix C_{t+1} which contains the pairwise-distance matrix between detections and tracklets coordinates at frame $t+1$ along with the generated cell state c and hidden state h at the previous frame t. The LSTM is then trained to output A_t^i a vector representing the likelihood probabilities of detections in frame t being similar to tracklet i. The second part of the tracker (RNN), will be discussed in section 5.3. It is interesting to know that they didn't include any appearance

features (except detections) in their algorithm and managed to achieve better than classical affinity methods (Kalman filter with the Hungarian algorithm). In fact, they managed to run their algorithm at 165 FPS on MOT15.

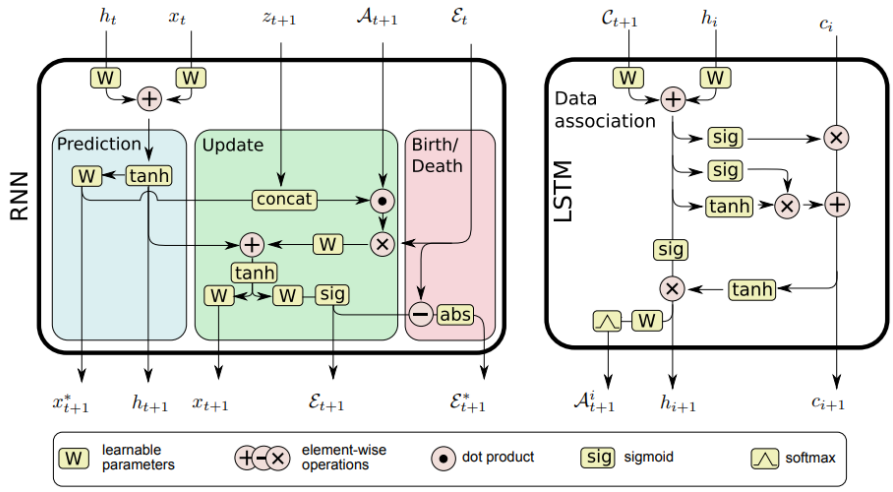


Figure 5.4. Summary of the tracker proposed in [99].

In contrast to the previous method (which utilizes distance between detections only), a more sophisticated one is proposed by [100]. A. Sadeghian et al. used 3 LSTMs networks (Figure 5.5), which are trained to capture appearance, motion, and interaction information. They all take the last t number of samples from a tracklet, and the $t+1$ element is a detection. The resulting vectors from the LSTMs are then fused and fed to a fully connected layer to output a similarity score. In other words, based on the tracklets' past motion, appearance, and interaction features, the network examines if detection could fit into that tracklet. A good statement is made in their work, which is that 80% of the MOT16/MOT17 occlusions last for six frames on average, and this is why they chose t to be six (will be used later to enhance the proposed tracker). They managed to reach the top rank on multiple benchmarks and increase the MOTA score by 18.6 compared to the previous approach [99] on MOT15.

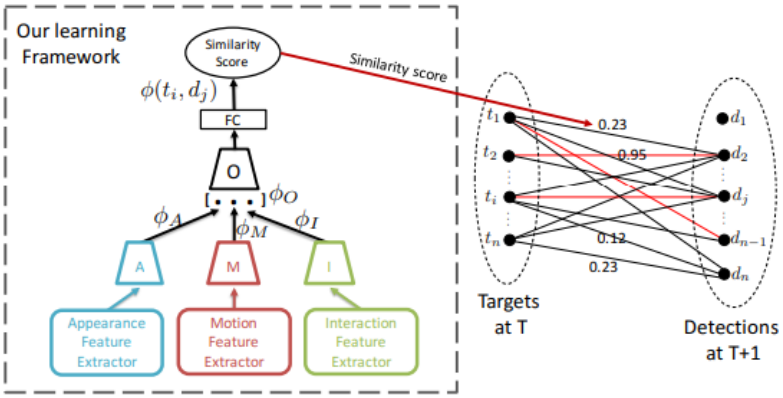


Figure 5.5. Summary of [100] end-to-end DL based affinity computation.

Similar to [99], [100]. [96], [97], [101] are some of the recent work that adapts recurrent neural networks for affinity and association (end-to-end approach). In [96], [97], a bilinear and CNN based LSTM topologies are used, respectively. Both topologies are trained on CNN generated features (e.x. ResNet50 [78] with top-layer removed). While [101] used CONV-LSTM layers trained on tracklet-detection pairs. More details on CONV-LSTM will be provided in the next chapter. All the beforementioned methods managed to achieve near top-ranked tracker performance ($MOTA \approx 52$) on the MOT16 benchmark.

5.2.2 Unsupervised Tracker

One of the unusual, totally different approach is proposed by S. Karthik et al. [83]. They proposed an unsupervised tracker, which is currently ranked one in the MOT20 dataset [102]. The algorithm, summarized in Figure 5.6, starts by collecting trajectories from a baseline method DeepSORT [76]. Then the Reid (ResNet-50) network is initialized to the number of classes equal to the noisy tracklets. ResNet-50 is trained to output affinity score for a bounding box being an element of a tracklet. At inference, the supervised method output is refined with their Reid model. I believe that this work is a paradigm shift in the filed of MOT because it reveals the

debate on whether we need to use sophisticated supervised DL based trackers or reconsider the power of the unsupervised approach.

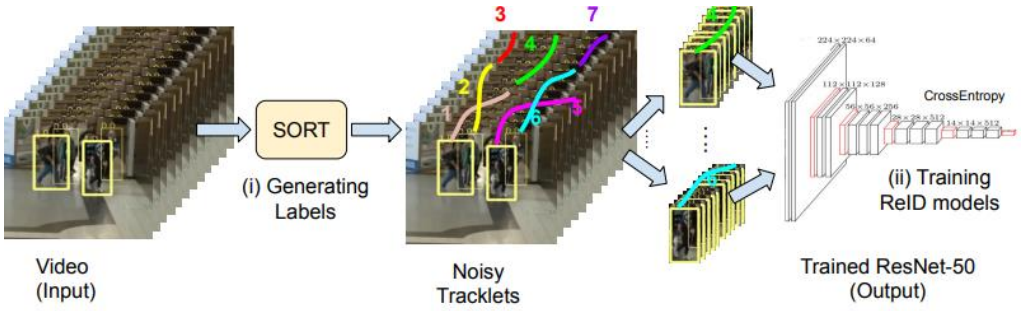


Figure 5.6. The tracker's pipeline proposed by [83].

5.3 Association Step

The Association step is the last step of a multiple object tracker. It is the task to classify the corresponding prediction for each target uniquely, and it is the most challenging component of MOT c. The conventional way to address the association issue is to find the minimum distance (e.x. cosine distance) between tracklets and detection from the affinity matrix as in DeepSORT [76], or by computing the spatial overlap (e.x IoU). However, the former approach fails if we are not tracking easy targets (i.e., targets are at a distance from each other), and the latter fails if the targets are occluded, or we encounter a high missing detection rate. This issue is addressed by exploiting the power of DL based features. In other words, the DL networks are used to improve the decisions made by classical methods.

The first work, to begin with, is [99], which part of it was discussed in section 5.2.1. The RNN network was trained to work as an end-to-end association and tracklets management system. All targets coordinate combined in a matrix x_t at a given frame t are fed to prediction gate (Figure 5.4) to output predicted states for all targets x_{t+1}^*. Note that the prediction gate does not use any knowledge about detections or LSTM affinity scores that is to force it to learn robust motion features. This prediction is then refined at the update gate, which makes use of available detections z_{t+1} at frame

$t+1$ and the LSTM output A_{t+1}^{i}. The update gate outputs ε_{t+1} which represent probabilities indicating for each tracklet how likely it is a real trajectory. Finally, the proposed data associations A_{t+1}^{i} by LSTM and the update gate output ε_{t+1} are fed to the Birth/Death gate shown in Figure 5.4 to decide whether to keep the associations or initialize new tracklets. Put merely, if a detection didn't match with a tracklet, the RNN layer will birth (initialize) a new trajectory. From their work, they reached the conclusion that LSTMs can learn a one-to-one assignment. Noting that this is one of the few trackers that are end-to-end DL-based trackers (except distance measurements for C matrix). However, the previous approach tends to fail when objects are highly overlapping. Some existing works tried to overcome this issue. Among those aspects, [100], [103]–[105] introduced more robust association strategies. For example, in [100] (Figure 5.5) and [103], a bipartite graph is formed, and the associations are done by minimizing the graph cost. While in [104], a separate head detection network is utilized, and another network scores detection-tracklet pairs through the fusing of the head and body coordinates. Similarly, Z. Lian et al. [105] employ a facial landmark localization network to match targets. i.e., ten facial features are collected (e.x. mouth, nose, eyes, etc..) from all targets, and then minimum cosine distance is used for association.

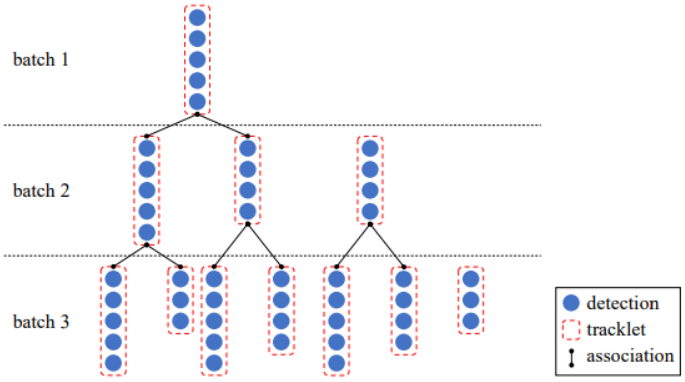


Figure 5.7. Tracklets association framework. It constructs a tree which is updated while tracking with a window of 3-5 frames [106]. A batch refers to the recent 3-5 consecutive frames detections associated together.

Y. Zhang et al. [106] proposed a different technique that is to make an association a dynamic task. The trajectories are split into groups of 3-5 frames referred to as batches in Figure 5.7. Then they are fed as pairs along with detections to what they call Appearance evaluation network (AEN) and Motion evaluation network (MEN) to output probabilities of detection being an element of that pair. Associations are performed at that frame according to the scores. Note that one batch can be assigned to other batches from the previous frame. However, processing the consecutive frame, the associations are pruned, and only one path is kept at the end of the tracking task. The main idea of the work is to give the tracker chance to fix bad associations by having a window (of size 3-5 blocks) to update the trajectories. Y. Zhang et al. managed to perform quite well in the MOT17 dataset, ranked fifth among the published online trackers.

CHAPTER 6

TOP ONLINE TRACKERS EVALUATION

In the previous chapters, we have discussed the trackers' pipeline and elements without examining their numerical performance and drawbacks. Therefore, in this chapter, we will start by analyzing top online performing methods on MOTChallenge and highlight their bottlenecks. In order to attack these problems, we chose Tracktor [74] as the baseline network to apply our novel solutions. Hence, in the second part of this chapter, we will illustrate the baseline method in more detail.

6.1 Analysis & Comparison

Table 6.1 below presents the ten top online trackers that have tested their algorithms on MOT17. The table summarizes their pipeline according to our categorization from chapter 5 and 4. Table 6.2 shows the numerical performance of the same methods from Table 6.1. All analyzed trackers use public detections except [74], which uses a trained Faster R-CNN model. If the public detections are used, the averaged performance is reported by utilizing all three MOT available detectors. The quantitive results are obtained from the public leaderboards on the MOTChallenge website. In other words, only the performance on the test set is reported, which is to eliminate the overfitting problem on the train set. It is important to note that the FPS metric will be neglected because every method experimented on different hardware setup and platform.

Table 6.1 The summary of the top online trackers on MOTChallenge 2017 in descending order according to their MOTA performance.

	Appearance Handling		**Grouping Step**	
	Object Detection	**Feature Extraction**	**Affinity**	**Association**
[74]	Faster R-CNN	ResNet-50	ResNet-50	TriNet [107]
[108]	External tracker		Deep Hungarian Net	
[109]	Public	Siamese-RPN and Inception-v4	IOU	Newton boosting decision tree
[110]	Public	Siamese-style Network	Affinity Sub-Net + Target Management	
[111]	Public	Modified ResNet	MatchingNet [112]	
[113]	Public	Spatial-Temporal Relation Networks to generate Bipartite Graph		Hungarian algorithm
[114]	Public	R-FCN with SqueezeNet as the backbone	Modified GoogLeNet	Minimum Euclidean distance
[115]	Public	Siamese-Net	Joint inference network	Hungarian algorithm

Table 6.2 The numerical performance of the top online trackers. MT and ML are in a percentage format. The arrows indicate low or high optimal values.

	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDSW ↓	Frag ↓
[74]	**56.3**	55.1	21.1	35.3	**8866**	**235449**	1987	3763
[108]	53.7	53.8	19.4	36.6	11731	247447	**1947**	4792
[109]	52.7	**57.9**	17.9	36.6	22512	241936	2167	7443
[110]	52.0	48.7	19.1	**33.4**	14138	253616	3072	5318
[111]	51.4	54.0	**21.2**	37.3	29051	243202	2118	**3072**
[113]	50.9	56.0	18.9	33.8	25295	249365	2397	9363
[114]	50.9	52.7	17.5	35.7	24069	250768	2474	5317
[115]	50.3	53.5	19.2	37.5	25479	252996	2192	3978

At first glance, we can notice that the complexity of the method does not necessarily imply achieving the top performance. As a matter of fact, the most straightforward approach among all is [74] with the highest MOTA score. [74] is one of the few approaches using private detections, confirming the fact we made in Chapter 4, section 4.1.3, that is acquiring high-quality detections is essential.

In addition to that, using the Siamese type association network to compute the discrimination between potential candidates as in [74] and [108] significantly reduced the number of identity switches. In both works, the raw RGB candidates are passed through a custom Siamese structured network where the embeddings are fused, and a correlation score is computed. The more examined candidates at one pass, the more computations are required, and a better performance might be achieved as in [108].

Adopting the LSTM layer within the grouping step shows a significant enhancement in the tracking period. K. YOON et al. utilized bidirectional LSTM in their proposed network to process the extracted features and built a conditional state for the targeted objects. For this reason, they managed to achieve the highest MT and the lowest

number of Frag. However, without satisfying the previous condition, i.e., good quality detections, they had the worst number of false positives.

A common problem among online trackers, which is substantially reflected in the MOTA score, is the high number of FNs, as we can see in Table 6.2. After careful inspection, we found that small bounding boxes with an area less than a threshold are one of the reasons for this issue. Most of the methods use public detections, which are collected from old detection models since 2017.

To cope with the above problems, we found that working on detections and an LSTM based end-to-end association solution may solve the problem. We adopted [74] as our baseline network to demonstrate the power of our proposals on the current top online tracker.

6.2 Baseline Tracker (Tracktor)

In 2019, P. Bergmann et al. [74] proposed a tracking-by-detection online tracker, which ranked 1st on three different MOTChallenge benchmarks at the time of writing this thesis. The main idea of the work is to convert a detection model into a tracker by using the tracking-by-regression paradigm.

[74] has one core element and two extensions:

1. **(core) Object detector:** The core element of [74]. Faster R-CNN with ResNet-50 and Feature Pyramid Networks (FPN) trained on the MOT17Det dataset.
2. **(extension) Motion model:** two motion models are used in two different scenarios, the first if the sequence has large camera movements, motion compensation technique is applied. Second, if the sequence has a low frame rate, a constant velocity assumption is adopted.
3. **(extension) Re-identification network:** TriNet architecture [107] is used to collect 128 features from deactivated tracklets and potential targets. Then A

CNN based Siamese network trained on the MOT17Det dataset is used for distance measurement. The minimum distance is chosen as a match.

6.2.1 Object Detector

To track objects, you need to predict objects in the next frame. For detection models to achieve this, a regression head should be integrated. Among regression-based detectors, Faster R-CNN is used due to its excellent performance on the MOTChallenge benchmark and other reasons elaborated in section 4.1.3. The regression head is trained to take the current frame and last bounding box coordinates to predict the location on the next frame instead of refining the detections on the current frame. We call this modification tracking-by-regression.

Tracktor uses a shallower backbone network (ResNet-50) with the Faster R-CNN. We will show later that using a more in-depth version (ResNet-101) is a better option. Especially with our calibration technique where robust features are required.

The authors also integrated the Feature Pyramid structure [116] to the model. Meaning, instead of utilizing the last convolutional layer for predictions only, a prediction can be made at any level by upsampling the previous feature map to match the size and add it to the current level output (Figure 6.1). This technique has proven to enhance the Faster R-CNN model's ability to detect small objects [116].

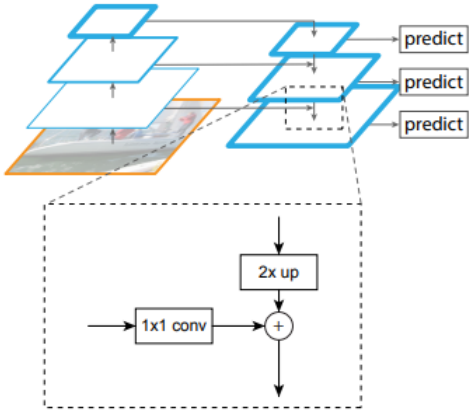


Figure 6.1. Faster R-CNN Feature Pyramid building block [116].

6.2.2 Motion Model

Conventionally, trackers assume that pedestrians' location changes slightly from frame to frame. Hence, no motion model is required. However, this assumption, according to P. Bergmann et al. [74], fails in two scenarios.

1. When the camera has large movements, and it is solved by using the Image Enhanced Correlation Coefficient (ECC) algorithm introduced in OpenCv3 and based on the paper [117]. ECC is an iterative algorithm that tries to find a warp matrix which deforms the grid of a frame to align it to its previous one.
2. When the video has a low frame rate, a massive displacement is expected from the targets. Hence, a constant velocity assumption (CVA) is used. Meaning, detections coordinates will be shifted by the difference of the last two coordinates (constant velocity).

[74] uses two thresholds to decide whether to deactivate the tracklet or not:

1. If the classification score s of the last bouning box $< \sigma_{active}$ the tracking is deactivated. The object is then assumed to be out of the scene or occluded by non-object (e.x. electric pole).
2. Non-Maximum Suppression (NMS) is applied between detections with IoU threshold λ_{active} to decide whether an object (i.e., pedestrian) is occluded by other objects.

Note That the motion model is applied to deactivated tracklets for possible reactivation after an occlusion scenario.

The next three pages contain the pseudocode of the [74] algorithm, NMS, and reidentification network. It is followed by Figure 6.3, showing the whole tracking pipeline.

Algorithm 1: Tracktor algorithm

Input Image list $\mathbf{I} = \{\boldsymbol{i_0}, \boldsymbol{i_1}, \ldots \boldsymbol{i_t}\}$ where $\boldsymbol{i_t}$ represent the frame $\boldsymbol{i}$ at time $\boldsymbol{t}$.

Output set of trajectories $\mathbf{T} = \{\boldsymbol{t_0}, \boldsymbol{t_1}, .., \boldsymbol{t_k}\}$ with $\boldsymbol{t_k} = \{\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t_1}}, \boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t_2}}, \ldots, \boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t_j}}\}$ where $\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t_j}}$ refer to the bounding box coordinates $\{\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{h}, \boldsymbol{w}\}$ of the object $\boldsymbol{k}$ at the frame $\boldsymbol{t_j}$.

Procedure $\mathbf{T} \leftarrow$ Tracktor($\mathbf{I}$)

$\mathbf{T}, \mathbf{T_{active}} \leftarrow \boldsymbol{\phi}$ // initialize two vectors to track tracklets and the active ones

for $\boldsymbol{i_t}$ in $\mathbf{I}$:

$\boldsymbol{B}$,$\boldsymbol{S}$ $\leftarrow \boldsymbol{\phi}$ // vectors to store bounding boxes (bboxes) and their scores

// update active tracklets by using F-RCNN regression head (check 1 2)

for $\boldsymbol{t_k}$ in $\mathbf{T}_{\boldsymbol{active}}$:

$\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t-1}} \leftarrow \boldsymbol{t_k}[-\mathbf{1}]$ // last bounding box

$\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t}}, \boldsymbol{s}^{\boldsymbol{k}}_{\boldsymbol{t}} \leftarrow$ detector.regression_head($\boldsymbol{i_t}$, $\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t-1}}$)

if $\boldsymbol{s}^{\boldsymbol{k}}_{\boldsymbol{t}} < \boldsymbol{\sigma_{active}}$:

$\mathbf{T}_{\boldsymbol{active}} \leftarrow \mathbf{T}_{\boldsymbol{active}} \setminus \{\boldsymbol{t_k}\}$ // liminate $\boldsymbol{t_k}$ from active tracklets

$\mathbf{T} \leftarrow \{\boldsymbol{t_k}\}$

else:

$\boldsymbol{B} \leftarrow \boldsymbol{B} + \{\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t}}\}$

$\mathbf{S} \leftarrow \mathbf{S} + \{\boldsymbol{s}^{\boldsymbol{k}}_{\boldsymbol{t}}\}$

B $\leftarrow \mathbf{NMS}(\boldsymbol{B}, \boldsymbol{S}, \boldsymbol{\lambda_{active}})$ // NMS removes all occluded bounding boxes

for $\boldsymbol{k}$, $\boldsymbol{t_k}$ in $\mathbf{T}_{\boldsymbol{active}}$:

if $\boldsymbol{k} \notin \boldsymbol{B}$:

$\mathbf{T}_{\boldsymbol{active}} \leftarrow \mathbf{T}_{\boldsymbol{active}} \setminus \{\boldsymbol{t_k}\}$ // liminate $\boldsymbol{t_k}$ from active tracklets

$\mathbf{T} \leftarrow \{\boldsymbol{t_k}\}$

for $\boldsymbol{t_k}$, $\boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t}}$ in zip($\mathbf{T}_{\boldsymbol{active}}$, $\boldsymbol{B}$) :

$\boldsymbol{t_k} \leftarrow \boldsymbol{t_k} + \boldsymbol{b}^{\boldsymbol{k}}_{\boldsymbol{t}}$ // add boxes to the corresponding trajectory

$\boldsymbol{D} \leftarrow$ detector.detection_head($\boldsymbol{i_t}$)

// remove detections that interesect with active tracklets

for $\boldsymbol{d_t}$ in $\boldsymbol{D}$:

for $\boldsymbol{b_t^k}$ in $\boldsymbol{B}$:

if $\boldsymbol{IoU}(\boldsymbol{d_t}, \boldsymbol{b_t^k}) \geq \mathbf{0.5}$:

$\boldsymbol{D} \leftarrow \boldsymbol{D}\ \{\boldsymbol{d_t}\}$ // liminate the overlapping detection

$\mathbf{T}, \mathbf{T}_{\boldsymbol{active}}, \boldsymbol{D} \leftarrow \boldsymbol{ReID}(\boldsymbol{T}, \mathbf{T}_{\boldsymbol{active}}, \boldsymbol{D})$ // Re-identification network

// Potential detections in $\boldsymbol{D}$ will be initialized as a new trajectories

for $\boldsymbol{d_t}$ in $\boldsymbol{D}$:

$\boldsymbol{t_k} \leftarrow \boldsymbol{\phi}$

$\boldsymbol{t_k} \leftarrow \boldsymbol{t_k}\ + \{\boldsymbol{d_t}\}$

$\mathbf{T}_{\boldsymbol{active}} \leftarrow \mathbf{T}_{\boldsymbol{active}} + \{\boldsymbol{t_k}\}$

// Return the final set of trajectories

$\mathbf{T} \leftarrow \mathbf{T} + \mathbf{T}_{\boldsymbol{active}}$

End procedure*

* If public detections are used, **detector.detection_head** will be replaced with a text file reader.

Algorithm 2: Non-Maximum Suppression

Input bbox list $\boldsymbol{B}$, classification scores $\boldsymbol{S}$ and $\boldsymbol{IoU}$ threshold $\boldsymbol{\lambda_{active}}$.

Output filtered bbox list $\boldsymbol{B}$ and the corresponding classification scores $\boldsymbol{S}$

Procedure B $\leftarrow \mathbf{NMS}(\boldsymbol{B}, \boldsymbol{S}, \boldsymbol{\lambda_{active}})$

$\mathbf{B} \leftarrow \boldsymbol{\phi}$

for $\boldsymbol{b_t^k}$ in $\boldsymbol{B}$:

for $\boldsymbol{b_t^m}$ in $\boldsymbol{B}$:

if $\boldsymbol{IoU}(\boldsymbol{b_t^k}, \boldsymbol{b_t^m}) > \boldsymbol{\lambda_{active}}$:

if $\boldsymbol{s_t^k} > \boldsymbol{s_t^m}$:

$\boldsymbol{B} \leftarrow \boldsymbol{B}/\{\boldsymbol{b_t^m}\}$ & $\boldsymbol{S} \leftarrow \boldsymbol{S}/\{\boldsymbol{s_t^m}\}$

else:

$\boldsymbol{B} \leftarrow \boldsymbol{B}/\{\boldsymbol{b_t^k}\}$ & $\boldsymbol{S} \leftarrow \boldsymbol{S}/\{\boldsymbol{s_t^k}\}$

$\boldsymbol{return\ B}$

End procedure

NMS (illustrated in the above pseudocode) is a technique used to filter out overlapping pedestrians. The reason it is required is that the object detection model tends to fail to detect overlapping objects and return multiple bounding boxes that combine them. An example is depicted in Figure 6.2.

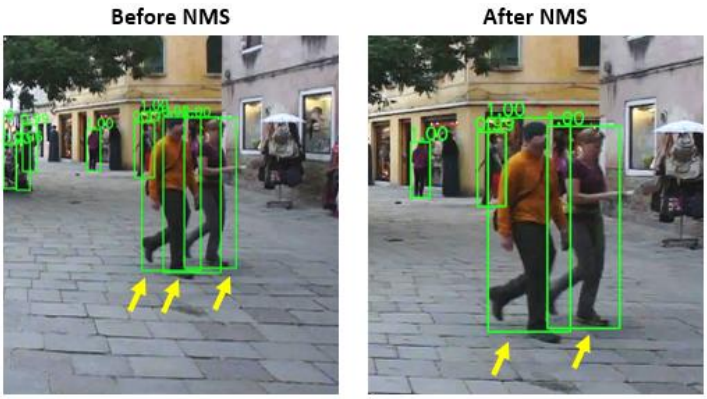


Figure 6.2. One the left: the middle detection combines both pedestrians. On the right: NMS filtering the overlapped detection.

Finally, the Siamese re-identification network tries to find a match between deactivated tracklets and potential detections :

Algorithm 3: Re-Identification Network

Input detection list $\boldsymbol{D}$, set of active and deactivate tracklets $\boldsymbol{T}$ and active tracklets $\mathbf{T}_{\boldsymbol{active}}$.

Output filtered $\boldsymbol{D}$ and updated $\boldsymbol{T}$, $\mathbf{T}_{\boldsymbol{active}}$ lists.

Procedure $\mathbf{T}, \mathbf{T}_{\boldsymbol{active}}, \boldsymbol{D} \leftarrow \boldsymbol{ReID}(\boldsymbol{T}, \mathbf{T}_{\boldsymbol{active}}, \boldsymbol{D})$

$\boldsymbol{T}_{\boldsymbol{deactive}} \leftarrow \boldsymbol{\phi}$

for $\boldsymbol{t}_{\boldsymbol{k}}$ in $\mathbf{T}$:

    if $\boldsymbol{t}_{\boldsymbol{k}} \notin \mathbf{T}_{\boldsymbol{active}}$:

        $\boldsymbol{T}_{\boldsymbol{deactive}} \leftarrow \boldsymbol{T}_{\boldsymbol{deactive}} + \{\boldsymbol{t}_{\boldsymbol{k}}\}$

for $\boldsymbol{t}_{\boldsymbol{k}}$ in $\boldsymbol{T}_{\boldsymbol{deactive}}$:

    $\boldsymbol{b}_{\boldsymbol{t-1}}^{\boldsymbol{k}} \leftarrow \boldsymbol{t}_{\boldsymbol{k}}[-1]$

    for $\boldsymbol{d}_{\boldsymbol{t}}$ in $\boldsymbol{D}$:

$\boldsymbol{s} \leftarrow$ SiameseCNN($\boldsymbol{b}_{\boldsymbol{t-1}}^{\boldsymbol{k}}$, $\boldsymbol{d}_{\boldsymbol{t}}$) // $\boldsymbol{s}$ is the match score

if $\boldsymbol{s} > \mathbf{0.5}$:

if $\boldsymbol{IoU}\left(\boldsymbol{b}_{\boldsymbol{t-1}}^{\boldsymbol{k}}, \boldsymbol{d}_{\boldsymbol{t}}\right)^{*} > \mathbf{0.6}$:

$\boldsymbol{t}_{\boldsymbol{k}} \leftarrow \boldsymbol{t}_{\boldsymbol{k}} + \{\boldsymbol{d}_{\boldsymbol{t}}\}$

$\mathbf{T}_{\boldsymbol{active}} \leftarrow \mathbf{T}_{\boldsymbol{active}} + \{\boldsymbol{t}_{\boldsymbol{k}}\}$

$\mathbf{T} \leftarrow \mathbf{T} / \{\boldsymbol{t}_{\boldsymbol{k}}\}$

$\boldsymbol{D} \leftarrow \boldsymbol{D} / \{\boldsymbol{d}_{\boldsymbol{t}}\}$

$\boldsymbol{return}$ $\mathbf{T}$, $\mathbf{T}_{\boldsymbol{active}}$, $\boldsymbol{D}$

End procedure

$*$ Because the motion model is applied for deactivated tracklets, then it is expected to have a high overlap with the potential detection.

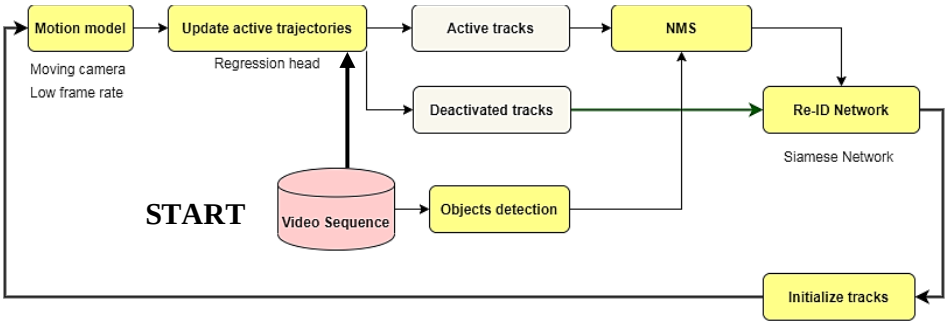


Figure 6.3. Summary of Tracktor tracking pipeline [74].

6.3 Baseline Tracker Performance Evaluation

To enhance the performance of [74], we need to understand its bottleneck. To achieve this, we run the tracker on the MOT17 dataset. In this section, we will discuss the tracking and timing performance of [74] on train and test sequences.

6.3.1 Performance on Test Sequences

The metrics are evaluated for every train sequence separately, and the average is taken as a final result. The values provided in Table 6.3 are collected from the MOTChallenge leaderboard, and it indicates the performance of the tracker on public detections only. i.e., the detections which are made publicly available by the challenge, as discussed in section 3.1.1.2. Because the MOTChallenge doesn't share the ground-truth labels for test sequences, we can neither run this evaluation in a local environment nor experiment with different backbones.

Table 6.3 Experiment results of [74] on test sequences (public detections). MT and ML are in a percentage format. The arrows indicate low or high optimal values.

Model	MOTA ↑	IDF1 ↑	MT ↑	ML ↓	FP ↓	FN ↓	IDSW ↓	Frag ↓
FRCNN	42.14	45.76	18.17	38.93	3918	83904	648	1461
DPM	41.9	45.04	16.69	42.04	2908	86275	635	1315
SDP	43.81	46.60	20.27	36.94	5375	77868	789	1835

6.3.2 Performance on Train Sequences

The experiments shown in Table 6.4. are done using both private and public detections in a local test environment. We used the authors' codes to retrain the network with Resnet-50 backbone and then experiment with the trained models in a private-run. We will choose the best results as our target for enhancement. Although the table below shows average results, a detailed sequence output is provided for the top performance approach in section 6.3.3.

Table 6.4 Experiment results in train sequences for both public and private detections. Total ground-truth detections are 112297.

Type	model	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
Private	FRCNN + FPN	**56.46**	**59.6**	**38.2**	**4.63**	13626	**33486**	1785	1919
Public	FRCNN	53.4	55.6	11.9	7.02	12784	38691	817	1768
	DPM	54.75	56.2	8.55	9.89	**8213**	42366	**233**	**854**
	SDP	51.4	54.3	13.4	**5.74**	14393	39373	837	2336

From Table 6.4, it is evident that the private training detection model is a better option than using public ones. The first to make this conclusion was A. Bewley et al. [69] when they proposed SORT, as discussed in section 4.1.3. we can also observe that integrating FPN reduced the number of FNs dramatically but comes with the cost of higher FPs (e.x. detecting manikin from MOT17-02 sequence). Moreover, the model is now capable of detecting small objects efficiently but with low stability. Meaning, the smaller the object, the more likely it will be lost or switched, causing a high number of fragmentations and IDSW, respectively. However, for the bigger ones, they are mostly tracked even under challenging occlusion scenarios. It is always a tradeoff between FN and FP. In the case of DPM, it managed to score the lowest FPs besides a massive FNs. Although we don't have enough details on how public models were trained to give a precise comment, it might be the overfitting issue that causes this behavior.

We have noticed that, on average, a slight 9 pixels intensity change inside a bounding box from frame to frame may lead to loss of the detection for the private model, especially for small-sized objects. Hence, utilizing a proper motion model is essential. From all our experiments and reviews, we found that The number of FNs mostly drives the MOTA score. In fact, FN is usually 2 to 10 times more than FP and IDSW combined.

6.3.3 Detailed Sequence Output

Since the private model performed the best in most of the metrics, we will try to enhance its performance in the next sections. The table below shows the evaluation report for every sequence in the training set. The bottlenecks are discussed on the next page.

Table 6.5 Detailed performance report using the private detection model.

Sequence	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
MOT17-04	69.62	68.8	19.28	9.64	3111	11243	96	165
MOT17-02	36.54	40.8	27.42	28.25	1054	9596	1141	263
MOT17-05	45.18	63.5	37.59	5.27	681	3031	80	330
MOT17-09	51.88	52.2	58.64	3.45	1899	618	45	61
MOT17-10	50.69	55.0	42.11	4.51	2668	3518	145	405
MOT17-11	41.16	54.4	39.10	4.10	1747	3567	238	116
MOT17-13	62.04	62.1	42.11	5.66	2466	1913	40	428
Overall	56.46	59.6	38.28	4.63	13626	33486	1785	1919

Unlike the other state-of-the-art trackers, [74] achieved better results in videos with dynamic camera motion such as MOT17-05, MOT17-13, MOT17-10. However, the tracking process was interrupted the most in these sequences. This can be seen from the high number of fragmentations. In fact, the MOT17-10 sequence was shot at a low light scene. Yet, thanks to the FPN, the model could barely lose a tracklet.

The MOT17-02 sequence scores the worst performance in terms of MOTA. This sequence has the highest number of false negatives and small objects per frame. Note that MOT17-04 has a higher FN value, but it with a double amount of frames. With careful inspection (Figure 6.4), we can see that misses are occurring in regions with a high small bounding boxes density.

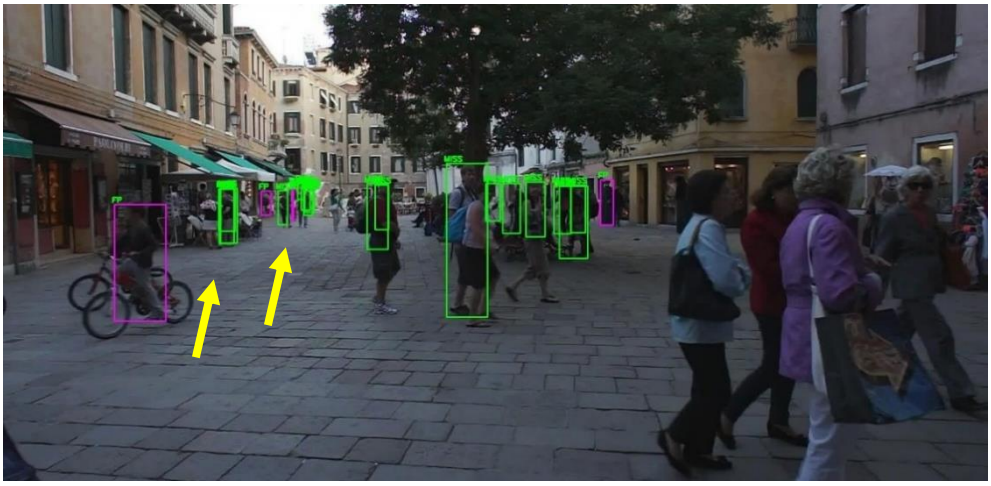


Figure 6.4. Frame 98 from MOT17-02 sequence. Green boxes are misses, purple ones are FP, and Arrows point to small regions that have high FNs.

6.3.4 Time Performance

Since the test is conducted locally, we can impartially examine the execution time performance of each experiment. Table 6.6 below indicates the time performance of [74] using the Faster R-CNN ResNet-50 backbone. Note that the Re-ID is roughly estimated by averaging association time for a given object, and it is highly dependent on the number of tracklets.

Table 6.6 Time performance with different backbones.

Model	Detection	NMS	Regression	Re-ID	Total time / frame
FRCNN + ResNet-50	730 ms	70 ms	550 ms	30 ms	1380 ms

CHAPTER 7

PROPOSED METHODS

After evaluating many DL based trackers and building a good background about the current approaches, we found it significant to contribute to this field by providing novel approaches to solve the bottlenecks illustrated in chapter 6. We went through several enhancement attempts. In this chapter, we will mention the major ones, why they failed or succeed. As a result of these experiments, we propose a novel technique to calibrate Faster R-CNN for a specific problem and a state-of-the-art real-time association network.

7.1 Yolo-v4

To keep up with the latest fast and accurate detection models and to cope with the problem of utilizing old public detections as elaborated in section 6.1, we replaced the detection model with Yolo-v4 [118] with slight modification. In April 2020, A. Bochkovskiy et al. published an updated version of Yolo-V3, which is discussed in section 4.1.1. The significant additions to the new model can be summarized in three points:

1. **Architecture**: a denser backbone with skip-connections, partial-connections, and adapting FPN.
    a. Skip-Connections: feeding one feature map to a different level than the consecutive one (Figure 7.1 - a).
    b. Partial-Connections: dividing feature maps to multiple branches, as shown in Figure 7.1 – b.

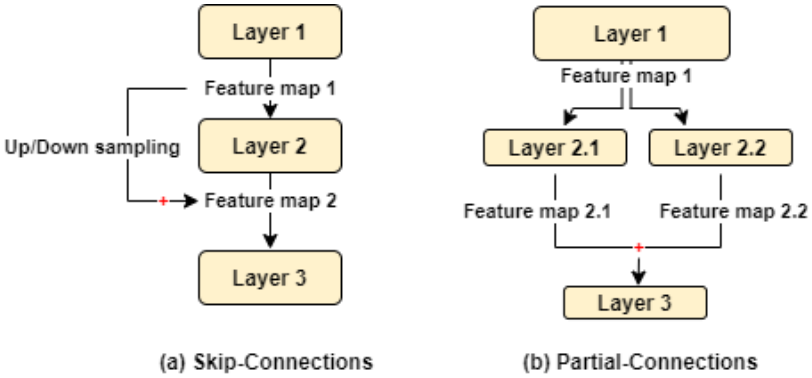


Figure 7.1. Example diagram for partial and skip connections.

2. **Bag of Freebies (BoF):** during training, new data augmentation techniques are used Namely. MixUp, CutMix, and Mosaic. They have proved that these techniques reduce the training loss much faster.
3. **Bag of Specials (BoS):** using the Mish activation function [119] to solve gradient vanishing problem with the increase of backbone network depth.

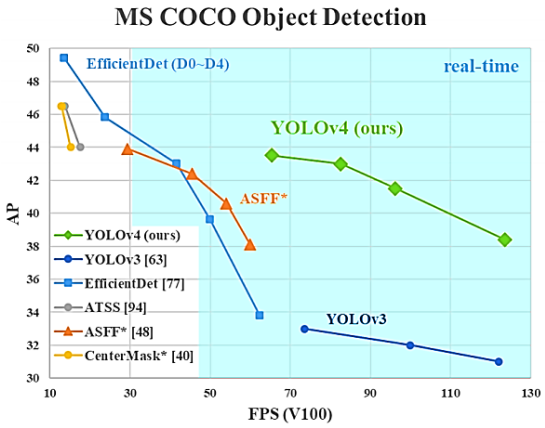


Figure 7.2. Comparison of Yolo-v4 to other popular detection models [118].

As can be seen from Figure 7.2, Yolo-v4 is 12% times faster than the previous version with a 10% higher AP. Interested readers may refer to [118] for in-depth review.

7.1.1 Training Phase

For training, we used AlexeyAB implementation [120] with Nvidia RTX 2080 GPU. We tuned the standard architecture to enhance small object detection. The best ones are reported in Table 7.1 below. Put merely, the maximum number of possible detections is increased to 250. Strides are halved to increase the number of scanned regions. Finally, convolutional layers are deepened to extract complex features.

Table 7.1 Best training configurations

Modified Line	Value
Last [yolo] layer	Max = 250
Lines 892 & 989	Stride = 2
Line 895	Layers = 23

We trained the network for 100,000 iterations on MOT17Det; each iteration includes four samples. The training process took around five days (13.85 iterations/minute). We utilized the weights checkpoint of the iteration 35,000 (with mAP 81.3%) that is to overcome the overfitting issue (Figure 7.3).

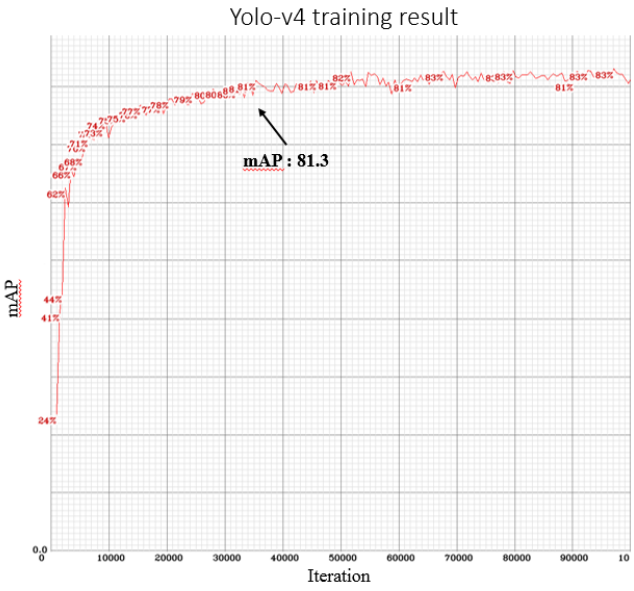


Figure 7.3. Yolo-v4 performance after training.

7.1.2 Inference Phase

In [74], we replaced the Faster R-CNN detection head with Yolo v-4 while keeping the pre-trained Faster R-CNN regression head. More importantly, we had to deal with a significant GPU challenge. Yolo uses the Darknet framework written in C [121] while the tracker is based on the Pytorch framework written in Python. It is tough to integrate two models from two different frameworks on a single GPU. Although it is not the scope of the thesis to discuss Nvidia GPU operations, we overcome this issue by forcing Pytroch and Darknet to lock distinct memory sections on the GPU. The first experiment failed to achieve decent performance, but with tuning confidence thresholds, we achieved better results.

Table 7.2 Yolo v-4 experiments result for the MOT17 dataset.

Exp	model	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
Exp 1	Yolo v-4	46.7	44.1	8.90	11.12	1052	48236	584	710
Exp 2	Yolo v-4	52.6	57.4	11.50	8.94	3421	33791	**374**	**401**
Exp 3	**Yolo v-4 + FRCNN**	**58.21**	59.23	**39.37**	12.21	**10117**	**36016**	792	1740
Baseline method		56.46	59.60	38.28	**4.63**	13626	**33486**	1785	1919

For the first experiment, we used [74] standard thresholds. i.e., 0.5 for both detection and regression filtering process and 0.3 for NMS. As can be seen from Table 7.2, the performance was deficient compared to previous public and private ones. The reason is that Yolo v-4 uses confidence threshold 0.1 for training, which is useful to boost the recall much faster, but it reduces the detection confidence scores at the same time. A sample frame detection is depicted below (Figure 7.4). After filtering, only four bounding boxes are left.

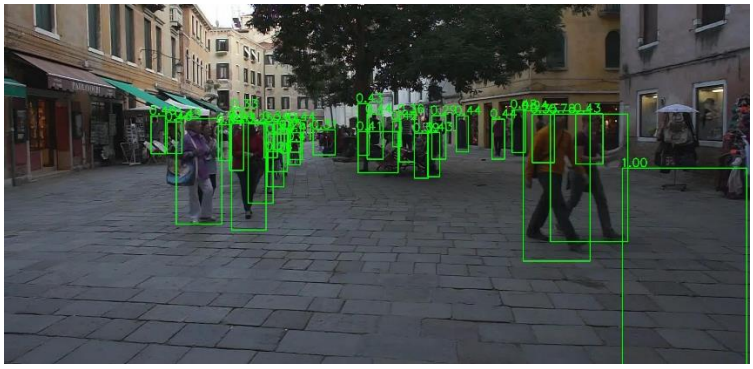


Figure 7.4. Sample Yolo-v4 detection output from sequence MOT17-02 frame 1.

7.1.2.1 Threshold Tuning

For the second experiment, we had to calibrate the confidence threshold to achieve optimal performance. Since MOTChallenge provides labels for the test sequences only, we took 10% of the frames from every video for the validation set. We run the detection model Yolo-V4 on the validation set with the detection threshold in the range [0.1,1.0] and 0.05 step size. The plot depicted in Figure 7.5 below shows that the highest ratio indicating high TPs and low FPs can be achieved with a threshold of 0.45. In experiment 2 (Table 7.2), we run the tracker with the tunned threshold. As expected, the number of missed detections is significantly reduced, and the tracking lasts for a longer time, which can be understood from the increase of IDF1 and MT metrics.

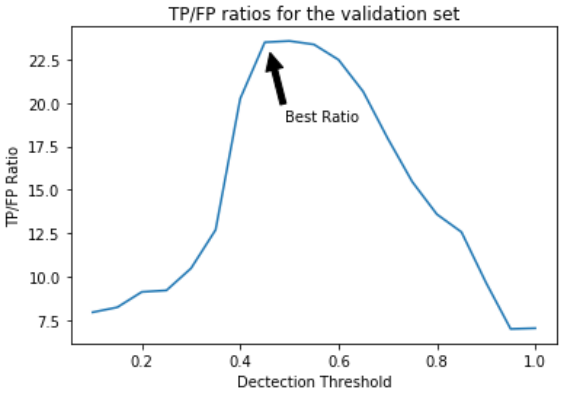


Figure 7.5. TP/FP ratios of Yolo-V4 detections on the validation set.

7.1.2.2 Ensembling Object Detectors

The final experiment was intended to enhance detection quality by integrating two detection models. Ensembling object detectors can be done in three ways, namely affirmative, consensus, and Unanimous [122].

- Affirmative: all collected detections from different detectors are concatenated and will be taken into consideration.
- Consensus: a detection is kept only if at least half of the detectors include it.
- Unanimous: all methods should agree on detection to be included.

Since we only have two detectors, both affirmative and consensus methods will provide the same results. Our aim from this experiment is to utilize the advantage of Yolo-V4 in detecting small objects. Hence, using Unanimous is not an option since FRCNN misses small objects. On the other hand, we tested concatenating all detections (affirmative or consensus method), and as expected, the number of FPs significantly increased because Yolo-V4 false positives are concentrated around small objects while it's the opposite for FRCNN. To overcome this issue, we use the conditioned consensus method. Meaning, only high confident FRCNN detections are concatenated to Yolo-V4 ones. With this approach, both small bounding boxes (from Yolo-V4) and large bounding boxes are included (from both Yolo-V4 & FRCNN). We can see from Table 7.2 that there is a slight improvement in the MOTA score with a 1.75% increase along with a lower number of FP and IDSW compared to the baseline approach. Example outputs are depicted in Figure 7.6. The objects which were not detected by the original approach are arrowed in (c). Small object detection is significantly improved.

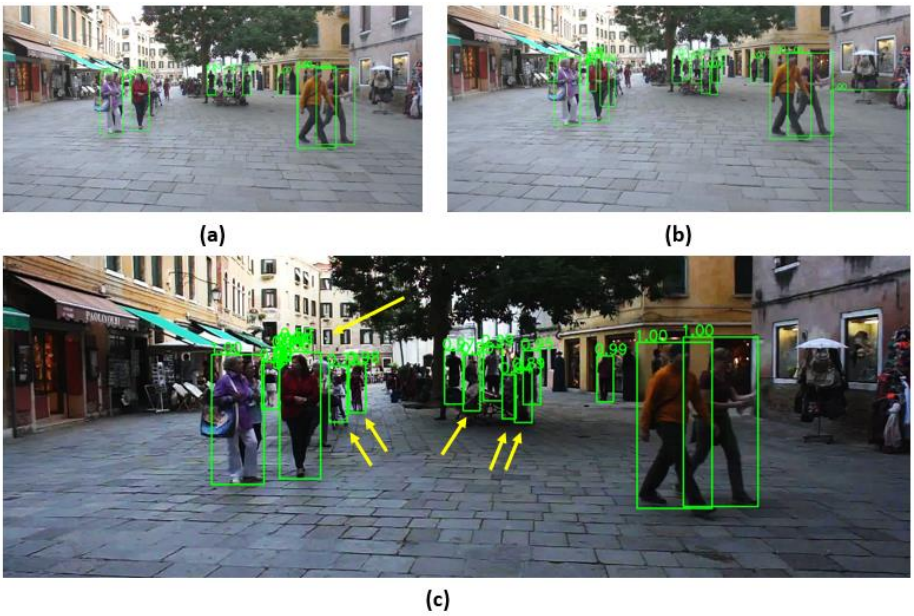


Figure 7.6. (a) original Faster R-CNN detection with a 0.5 filtering threshold. (b) Yolo-v4 experiment 2 example output. (c) Integrated Yolo + Faster R-CNN output.

7.2 Calibrated Faster R-CNN

We were inspired by C. Eggert et al. [123] study on how it is significant to modify the Faster R-CNN scales and anchors for better performance. Calibrating detection models will help with the detection of small objects, which a common problem discussed in chapter 6. C. Eggert et al. proposed a relationship to determine the optimal backbone feature map size for better small object detection on the FlickrLogos dataset [124]. However, their approach increases the computational requirements because of the increase in the search region (anchor points). The heuristic we are proposing eliminate redundant Region Proposal Network (RPN) search regions and decrease the number of bounding boxes required for every anchor. However, our approach requires a rich training dataset that is similar to the testing one.

The method simply starts by forming two histogram plots for all bounding boxes in all training sequences separately. Equations 7.1 and 7.2 are used to compute the Aspect Ratio (AR) and Anchor Scale (AS), respectively, for a given bounding box b and sequence s. All values are accumulated, and a histogram is formed as in Figure 7.6.

$$AR_b^s = \frac{b(0) - b(2)}{b(1) - b(2)} \tag{7.1}$$

$$AS_b^s = \sqrt{\big(b(0) - b(2)\big) \times \big(b(1) - b(2)\big)} \tag{7.2}$$

Assuming that the bounding box b is in $[x_{min}, y_{min}, x_{min}, y_{min}]$ Format.

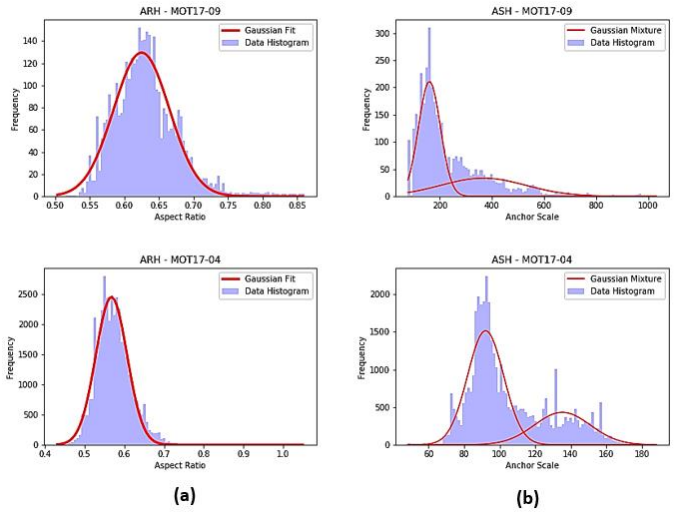


Figure 7.7. Sample AR and AS histograms for both MOT17-09 & 04 sequences.

Next, a Gaussian is fit to each AR plot while two Gaussian mixtures are fit to AS ones. The reason is that for AS, we want to cover all summits to target the majority of ground-truth bounding boxes better, as can be observed from the sample histograms depicted in Figure 7.6. Finally, we pick the mean from the fits for both AR and AS. For MOTChallenge, the mean of AR was in the range [5.95,6.05]. To choose the optimal value within this range, we sum the frequencies corresponding to every possible value in the range. Then, we choose the mean, which points to the highest frequency sum to target the majority, which is 0.6 for the MOTChallenge

dataset. The same steps apply to AS histograms. Table 7.3 shows both the standard Faster R-CNN values and what we claim to be a better choice for the MOT17 dataset.

Table 7.3 The standard and our proposed AR and AS for the Faster R-CNN model.

	Anchor Scales	Aspect Ratios
Standard	[32,64,128,256,512]	[0.5,1.0,2.0]
Ours	[40,90,150,280,600]	[0.6,1.0]

We reduced the number of possible search anchors to 10 instead of 15, which means the inference time will slightly decrease. For a fair comparison, we utilized the same training optimizer (SGD) and initial weights from the previous model.

Table 7.4 Comparing training development of our and the standard model.

Model	Epoch	TP	FP
Original	3	72939	111441
	9	75432	77272
	18	76243	68231
	27	**76719**	**63910**
Ours	3	73417	89242
	9	75828	54558
	18	76417	48738
	27	**78752**	**43320**

Every three epochs, we save the weights and evaluate that model on all sequences. For TP, we use IoU of 0.5; otherwise, the detection is FP. We trained both networks for 27 epochs because, beyond this epoch, the model starts to overfit. From Table 7.4, we observe that our approach achieved around 67% and 3%, lower FPs, and higher TPs respectfully. Similarly, for tracking, our trained model made 4.97% better MOTA and lower FP, FN, and Frag. Meaning, the quality of detections is higher, and the objects are tracked for a longer time. Note that the detection time by frame is reduced to 493.5 ms, which is 32.4% less than the previous one.

Table 7.5 Performance comparison between calibrated and non-calibrated FRCNN.

model	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
Calibrated FRCNN	**61.43**	**61.85**	**42.64**	7.50	**12513**	**30014**	**782**	**1637**
Non-calibrated FRCNN	56.46	59.60	38.28	**4.63**	13626	33486	1785	1919

7.2.1 The Calibration Technique with other Datasets

Our calibration technique is not limited to MOTChallenge only. It can be used whenever we want to have a low miss rate for the majority of ground-truth bounding boxes. Note that this technique may not necessarily improve small object detection unless they form the greater part of the training dataset. As an example, Figure 7.8 depicts three AR plots for three sequences obtained from the UA-DETRAC dataset [46], which consists of real-world traffic scenes with rich annotations, including vehicle bounding boxes. Following the steps from section 7.2, we found that the mean takes the range [1.0,2.0]. Unlike the MOTChallenge, where a single point was chosen, two aspect ratios stand out among others because of their high and close frequency sum. Hence, a suitable choice will be the scale points 1.0 and 2.0. Note that the aspect ratios in the UA-DETRAC dataset are greater than or equal to one because, unlike pedestrians, the width of a vehicle is greater than its height.

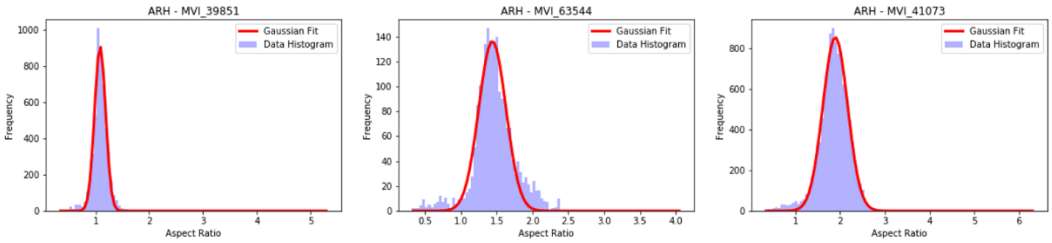


Figure 7.8. Sample AR plots for three UA-DETRAC [46] videos.

7.2.2 Every Other Frame

As pointed previously in section 6.2.1, the backbone network plays a vital role in features robustness. ResNet-101 [78] is the deeper architecture of ResNet-50 with better performance on the PASCAL VOC dataset [49]. However, utilizing a more in-depth architecture will rise detection time. To cope with this tradeoff, we trained Faster RCNN with ResNet-101 and FPN with the same training setup and calibrations from Table 7.3. Not to effect tracking time, we collect detections Every Other Frame (EOF). Meaning only odd frames will be fed to the detector. Note that the motion model is still used in all frames. Thresholds are calibrated to 0.40 and 0.45 for detection and regression, respectively. Table 7.6 illustrates how it is possible to achieve at most 7.72% better MOTA with our Calibration and Every Other Frame technique. It can be inferred from Table 7.7 that, with calibrated Faster R-CNN, the inference time can be reduced by 32.13%, compared to the standard one.

Table 7.6 Performance evaluation of the calibrated FRCNN with EOF.

model	EOF	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
(a)Calibrated FRCNN + ResNet-50	No	61.43	61.85	42.64	7.5	12513	30014	782	1637
	Yes	56.68	56.10	22.75	11.2	10740	37366	540	318
(b)Calibrated FRCNN + ResNet-101	No	64.18	68.10	40.38	9.9	4839	35023	368	557
	Yes	59.70	60.1	39.74	12.6	9284	35184	802	960

Table 7.7 Single frame inference time performance (in milliseconds).

Model	Detection	Effective detection	NMS	Regression	ReID	Total
Standard FRCNN + ResNet-50	730	730	70	550	30	1380
Calibrated FRCNN + ResNet-50	573	286.5				936.5
Calibrated FRCNN + ResNet-101	1086	543		731		1374

7.3 ConvLSTM2D based Siamese (CLS) Network

As part of our enhancement process, we studied the effectiveness of the [74] re-identification (REID) approach. REID and association tasks are two sides of the same coin. Association tries to match new detections with active trackers while REID functions the same but including deactivated tracklets too. Successful object REID demands robust features, especially to variations in orientation, illumination, and occlusion. For this reason, most of the state-of-the-art trackers use deep learning networks. Either by collecting DL based features and then find the object with the minimum distance [sources], or having an end-to-end network with a range-limited output. [74] adopted the latter approach, and it uses the TriNet architecture [107]. In this section, we will evaluate TriNet's performance on the MOTChallenge and propose our ConvLSTM2D based Siamese Network (CLSN).

7.3.1 TriNet

TriNet was proposed by A. Hermans et al. [107] to show the potency in using Triplet Loss to train re-identification networks and how it can outperform the other state-of-the-art approaches. A summary of the Triplet Loss and their proposal can be found in section 7.3.1.1.

Put merely, TriNet is a modified version of a pre-trained ResNet-50 [78]. They concatenated instead of the last layer, two fully connected layers—the first with 1024 units followed by 128 units one. The network has around 25M trainable parameters. In tracking context, TriNet takes as an input the latest four detections from every trajectory to output 128 feature vector. The vector is stored for ten frames and is compared before initializing any new tracklets with all active and deactivate targets (Chapter 6 – Algorithm 3).

7.3.1.1 Triplet Loss

Triplet loss is a loss function used to train neural networks on similarity problems. In other words, the NN will learn how to map specific class to euclidean space where the distance corresponds to the similarity measurement. Triplet loss was first used by F. Schroff et al. [125] to train FaceNet to recognize similar faces. Equation 7.3 provides the mathematical formulation or triplet loss

$$\sum_{i}^{M} \left[\left\| N(x_i^r) - N\left(x_i^p\right) \right\|_2^2 - \| N(x_i^a) - N(x_i^n) \|_2^2 + \beta \right] \tag{7.3}$$

Where x_i represents the ith training sample, which includes a reference (r), positive (p), and negative (n) samples. All samples are fed to the neural network N to collect three feature vectors. The aim is to minimize the above equation, which implicitly means minimize the distance between reference-positive samples. Simultaneously, maximize the reference-negative one. Note that β is a bias term and acts as a threshold. Figure 7.9 depict an example of how triplet loss is used to train CNNs. Note that CNNs can either be shared weights as in FaceNet or separate networks such as Siamese.

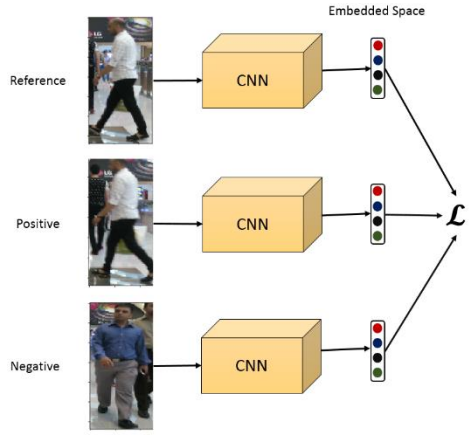


Figure 7.9. Sample training cycle where input x is fed to shared weights CNNs and the resultant features are used to compute the triplet loss $\mathcal{L}$.

7.3.2 Dataset Formation

A plain of REID dataset has been published in the last few years, such as CUHK03, Market1501, and MARS datasets. However, we found it best to create our custom-dataset generator. Since we are targeting MOTChallenge, we need to train the network on the same class and scenarios. i.e., samples generated from the MOT17 [7] dataset. Moreover, controlling the training samples will allow us to understand the behavior of the proposed network better.

MOT17 dataset has 1638 pedestrian trajectories with 336891 bounding boxes. A single training sample can either be a positive or negative anchor. A positive one contains bounding boxes from the same trajectory indicating the same object where the output of the network should be one, while the negative includes different tracklets. To better generate controllable training samples, we introduce three characteristics to our dataset. For better visualization, check Figure 7.10 below.

1. **Visibility factor:** it is a value between 0 and 1 to indicates how visible the object (Zero means the object is totally occluded). It is computed by measuring the overlap between an object and the surrounding bounding boxes. MOTChallenge already provides these values, and we will use it to control which samples to emphasize on while training.
2. **Anchors Time-Steps:** It determines how many samples per anchor should be used. I.e., if it is three, a negative anchor will have two positive samples and another from a different tracklet. Note that since we are trying to match a tracklet to potential detection, we can't train on an equal number of positive and negative samples. Only the last sample is allowed to be positive or negative (i.e., determines anchor type).
3. **Samples Time-Steps:** It resolves the gap between positive samples. As an example, In a positive anchor where samples and anchors time-steps are two. We will have two samples taken from the same tracklet but with a gap of two frames.

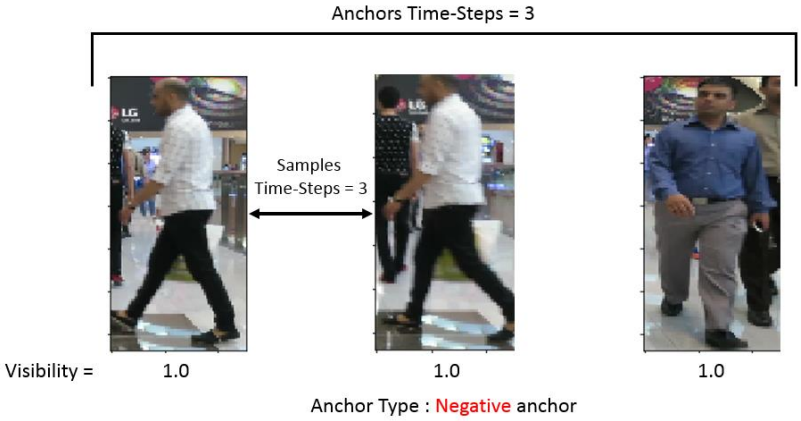


Figure 7.10. A training sample was generated from the MOT17 dataset. A negative anchor with three samples, all are totally visible and a gap of 3 frames between the first two samples.

7.3.3 CLSN Architecture

CLSN, as the name implies, is based on the ConvLSTM2D layer. This layer (Figure 7.11) is a Tensorflow implementation of X. Shi et al. work [126]. Similar to the beforementioned LSTM (section 2.5.1), except the matrix multiplications are exchanged with convolution operation at each gate. Every sample in the anchor is fed to the first convolution gate to generate a 1D array (feature vector). Then, the resultant vectors are either combined and form one state cell if the "return_sequences (RS)" option is set to True or every sample will have its own state.

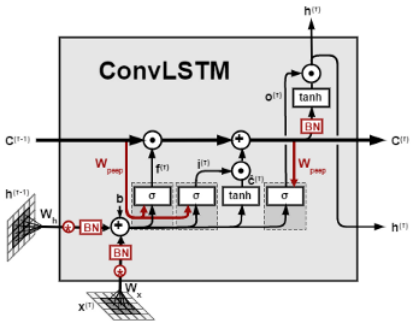


Figure 7.11. ConvLSTM2D layer architecture.

ConvLSTM2D was utilized in many works [72], [127], [128]. But up to our knowledge, it was not used previously in associating or REID multiple objects in tracking context. We aimed to use the minimum number of layers while preserving high accuracy. Hence we adopted Input Stacking topology (Figure 4.8 - c). We will refer to the number of ConvLSTM2D and the first Dense layer filters by x , anchors, and samples time-steps as a and s respectivly.

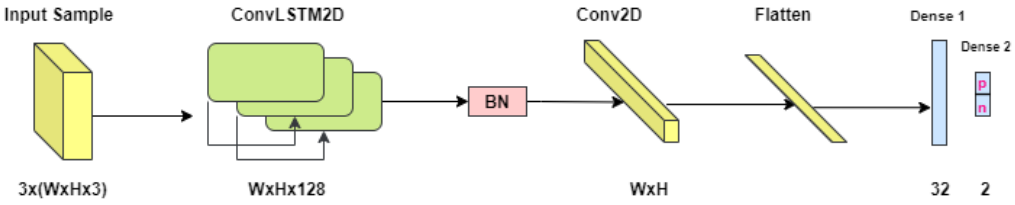


Figure 7.12. The graphical representation of the proposed CLSN.

Table 7.8 CLSN architecture details for $x = 128, a = 3$, and $s = 1$.

Layer	Arguments	Input Shape	Output Shape	Param.
Input Layer	-	3, 128, 64, 3	-	-
ConvLSTM2D	128 Filter: size 3×3 Padding "same" RS : False	3, 128, 64, 3	BS, 128, 64, 128	604160
Batch Norm	-	128, 64, 128	BS, 128, 64, 128	512
Conv2D	1 Filter: size 3·3 Padding "same" Sigmoid activation	3, 128, 64, 3	BS, 128, 64, 1	1153
Flatten	-	128, 64, 1	8192	-
Dense	128 units Relu activation	8192	128	1048704
DropOut	0.25 percent	-	-	-
Dense	2 units	128	2	258

* BS: Batch Size

The RGB stacked inputs are fed to the ConvLSTM2D layer to generate a three-dimensional state. Batch normalization is then applied to the output, as S. Ioffe et al. [129] suggested, to rescale all values along the third dimension. Hence the distribution of the weight is now shared between all time-steps (samples). Inspired by InceptionV3 [130], a single 2-D kernel is applied to reduce dimensionally without effecting the representational power. Finally, the resultant 2-D feature map is flattened and fed to two Dense layers for classification. A Dropout is used before the last classification layer to prevent independent learning among neurons [131]. Similar to ResNet and other classification networks, the last two layers can be eliminated or modified to fit into other applications such as the backbone of object detection networks. The overall graphical representation of CLSN is depicted in Figure 7.12.

After a few tests and calibration sessions, we found the following relations for better performance:

$$Sample\ Width = 16 \times k_1 \tag{7.3}$$

$$Sample\ Height = 2 \times Sample\ Width \tag{7.4}$$

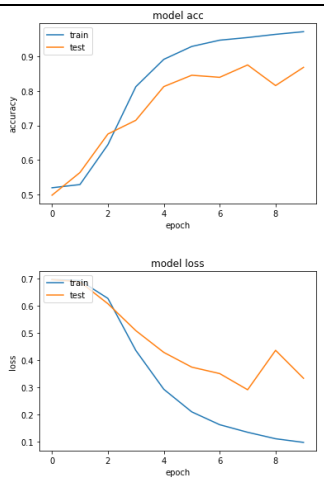
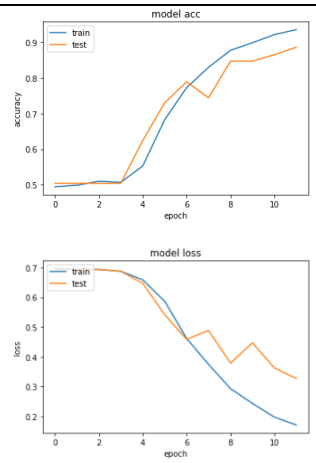
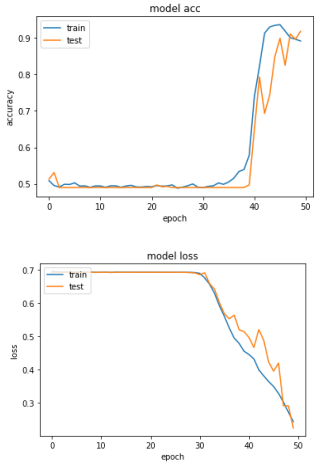
$$x = 32 \times k_2 \tag{7.5}$$

Where k_i is an integer in the range [1,4].

7.3.4 Training Phase

We generated 12, 4, 9 thousand random anchors for training, validation, and testing, respectively. The uniform distribution is used to handle the imbalance in the dataset, That is by keeping the frequency of positive-negative anchors and the high-low visibility reasonable. Data augmentations are applied while training to the training set. Namely, rotation, flip, hue, saturation, exposure augmentations are used. Training is performed on Nvidia GeForce RTX 2080 Ti and took a minimum 3947 s for $k_1 = k_2 = 1$ and maximum 6955 s for $k_1 = k_2 = 4$.

Table 7.9 Successful training sessions.

K values	Conf Matrix	Plots (top: accuracy, bottom: loss)	Time/sample
$k_1 = 2$ $k_2 = 4$	<table><tr><th></th><th>P**</th><th>N</th></tr><tr><td>P</td><td>3903</td><td>846</td></tr><tr><td>N</td><td>497</td><td>3754</td></tr></table>		18 ms
$k_1 = 2$ $k_2 = 2$	<table><tr><th></th><th>P</th><th>N</th></tr><tr><td>P</td><td>3649</td><td>388</td></tr><tr><td>N</td><td>993</td><td>3970</td></tr></table>		11 ms
$k_1 = 4$ **$k_2 = 1$**	<table><tr><th></th><th>P</th><th>N</th></tr><tr><td>P</td><td>4140</td><td>536</td></tr><tr><td>N</td><td>319</td><td>4005</td></tr></table>		**5 ms**

** P/N positive/Negative anchors.

Table 7.10 Training configuration for CLSN.

Option	Comment
Loss function	categorical_crossentropy
Optimizer	Adam Learning rate $= 0.0001$ $\beta_1 = 0.9$ $\beta_2 = 0.999$ $\varepsilon = 10^{-8}$
Labels	One-hot encoding [P,N]
Stop criteria	validation_loss rate < 0.02

Table 7.9 shows a few successful training sessions. The training is done on the TensorFlow framework with the settings from Table 7.10. Usually, for better data parallelization, the resultant gradient from different mini-batches is accumulated for a couple of iterations before updating the weights ([132] for more details). However, In our CLSN architecture, we have noticed similar behavior without explicitly addressing it in our implementation (Table 7.9 last experiment – model acc). It is still not clear if the CUDA implementation of Tensorflow is adopting this technique while training automatically. But what is essential is that the network should be carefully trained by allowing a more comprehensive number of epochs.

7.3.5 Inference Phase

The best Recall, Precision, and inference time is achieved with k_1 and k_2 , 4 and 1 respectively. With this implementation, we can re-identify eight objects in a real-time manner, which is six times faster than the previous approach (30 ms). To show the robustness of our network, below, we provide the confusion matrix for TriNet on the same test set. TriNet achieved around 10% and 7.9% lower positive and negative anchors accuracy, respectively.

Table 7.11 TriNet output results on our test set.

	P	N
P	3747	694
N	874	3685

To integrate our network with [74], we followed Algorithm 3 steps, except SiameseCNN is replaced with CLSN. For the detection step, we used the calibrated FRCNN with both shallow and deeper backend networks for inference (not including Yolo-V4). The matching loop is accelerated by narrowing down the search region. That is by starting with the spatially close elements.

Figure 7.13. Samples from inspection mode output before and after integrating CLSN. Arrowed objects in (b – the baseline method) are correctly tracked with CLSN.

Table 7.12 Tracking results before and after integrating CLSN.

Model	CLSN	MOTA	IDF1	MT	ML	FP	FN	IDSW	Frag
Calibrated FRCNN + ResNet-50	No	61.43	61.85	42.64	7.50	12513	30014	782	1637
	Yes	62.04	62.01	49.20	7.21	**12080**	30014	**535**	**1214**
Calibrated FRCNN + ResNet-101	No	64.18	68.10	40.38	9.94	4839	35023	368	557
	Yes	64.31	68.10	43.00	8.67	**4765**	35023	**294**	**471**
Baseline method	No	56.46	59.60	38.28	4.63	13626	33486	1785	1919

Table 7.12 shows the inference results of CLSN with our calibrated models. As expected, the same mistakes are made by the detector. i.e., the detector will still miss the same number of ground-truth objects (FN). However, we can observe the power of our CLSN network in the reduced number of IDSW (approximately improved by 30%). Simultaneously, fragmentations and FPs are reduced, indicating a more extended tracking period (stability) and a higher number of MT objects. Figure 7.13 shows a sample frame where our CLSN could reidentify a few challenging objects while TriNet failed.

CHAPTER 8

CONCLUSION

Multiple object tracking is an essential issue in the machine vision field. An object detection model or single object tracker can't be used directly for this problem. Many challenges arise with MOT, such as a high-density scene with intra-class variations. Yet, deep learning-based trackers have shown a remarkable performance over the years. In this thesis, we focused on the recent trends of this field, especially online trackers from MOTChallenge. After providing a brief summary for DL and MOTChallenge in chapter 1 and respectively, we have addressed multiple object trackers by categorizing them into two categories with two subcategories on each. After the intensive review and building a solid knowledge about top trackers, we proposed a novel technique for calibrating detection models and a network for the REID task.

8.1 DL based Multiple Object Trackers

Two steps can characterize DL based trackers. The first is appearance handling, where the tracker aims to collect robust invariant features from a specific class such as pedestrians. We consider this step to be the most vital one in the whole tracking pipeline. Detection and feature extraction determine the number of false positives and negatives which, by return, dominate the MOTA metric. After inspecting a lot of online trackers, we found that using the Faster R-CNN model for MOTChallenge provides high-quality detections. We saw a lot of work that uses classification networks to collect features from detection. But we believe reducing the overall complexity is important for better inference time; one may consider using a detention heatmap with a single-shot detector or the backbone network with two-stage ones. Finally, we addressed the grouping stage that is concerned with how to match

tracklets with potential detections. It is still challenging to find a DL based grouping algorithm that can run faster than a classical one (e.x. euclidean distance). But Siamese based architecture has shown to be promising by yielding high accuracy with low computational cost, in other words, combining feature extraction and affinity step. In this thesis, we discussed an end-to-end architecture for appearance handling or association step separately. However, we believe that the current development of MOTChallenge trackers opens up the way for an end-to-end MOT network.

8.2 Proposed Enhancements

After we have built a proper belief about DL based trackers, the second part of this thesis focused on our contribution to this field. The goal was to investigate the bottleneck of the beforementioned approaches and present possible improvement rooms. As an example but not limited to, we validated our enhancement techniques on Tracktor [74]. The first contribution of which is calibrating Faster R-CNN anchors and scales. Up to our knowledge, no work adopted this technique in the tracking context with dynamic scales and anchors range. Our strategy proved to strengthen the overall detection quality, even with skipping odd frames. As a result, the overall tracking performance was raised. Finally, we proposed a novel REID network, which we called it CLSN. The network architecture revealed the promising power of the ConvLSTM2D layer, which was disqualified unfairly from the MOT research community. Before ending the thesis, we must highlight the need to perform an ablation study to validate the robustness of CLSN and deploy it in different tasks such as classification, feature extraction, or single object tracking. Our final remark is that the MOT field is rapidly growing, and the focus now is on generic object tracking, which is the new orientation of MOTChallenge (ATO Challenge). Tracking is now extended to multiple classed objects tracking, which opens a broader opportunity again for classical methods to be hybrid with DL ones.

REFERENCES

[1] X. Fu, K. Zhang, C. Wang, and C. Fan, "Multiple player tracking in basketball court videos," in *Journal of Real-Time Image Processing*, 2020, pp. 1–18.

[2] V. H. Sridhar, D. G. Roche, and S. Gingins, "Tracktor: Image-based automated tracking of animal movement and behaviour," *Methods Ecol. Evol.*, vol. 10, no. 6, pp. 815–820, Jun. 2019.

[3] Y. Mao, L. Han, and Z. Yin, "Cell mitosis event analysis in phase contrast microscopy images using deep learning," *Med. Image Anal.*, vol. 57, pp. 32–43, Oct. 2019.

[4] Y. Mao and Z. Yin, "Two-stream bidirectional long short-term memory for mitosis event detection and stage localization in phase-contrast microscopy images," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10434 LNCS, pp. 56–64.

[5] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Commun. ACM*, vol. 60, no. 6, 2017.

[6] "Multi-object tracking with dlib - PyImageSearch." [Online]. Available: https://www.pyimagesearch.com/2018/10/29/multi-object-tracking-with-dlib/. [Accessed: 07-Sep-2020].

[7] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, "MOT16: A Benchmark for Multi-Object Tracking," Mar. 2016.

[8] P. Dendorfer *et al.*, "MOT20: A benchmark for multi object tracking in crowded scenes," Mar. 2020.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press,

2016.

[10] S. Skansi, *Introduction to deep learning: From Logical Calculus to Artificial Intelligence*, vol. 114, no. 6. 2018.

[11] "Machine Learning is Fun! - Adam Geitgey - Medium." [Online]. Available: https://medium.com/@ageitgey/machine-learning-is-fun-80ea3ec3c471. [Accessed: 31-May-2020].

[12] B. Uzkent, C. Yeh, and S. Ermon, "Efficient Object Detection in Large Images using Deep Reinforcement Learning," pp. 1813–1822, Dec. 2019.

[13] E. Crawford and J. Pineau, "Spatially Invariant Unsupervised Object Detection with Convolutional Neural Networks," *Proc. AAAI Conf. Artif. Intell.*, vol. 33, no. 01, pp. 3412–3420, Jul. 2019.

[14] "Artificial intelligence – IBM Developer." [Online]. Available: https://developer.ibm.com/technologies/artificial-intelligence/. [Accessed: 31-May-2020].

[15] N. M. Nawi, W. H. Atomi, and M. Z. Rehman, "The Effect of Data Pre-processing on Optimized Training of Artificial Neural Networks," *Procedia Technol.*, vol. 11, pp. 32–39, Jan. 2013.

[16] "The Neuron." [Online]. Available: https://www.brainfacts.org/brain-anatomy-and-function/anatomy/2012/the-neuron. [Accessed: 31-May-2020].

[17] Y. Bengio, P. Simard, and P. Frasconi, "Learning Long-Term Dependencies with Gradient Descent is Difficult," *IEEE Trans. Neural Networks*, vol. 5, no. 2, pp. 157–166, 1994.

[18] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[19] D. Mishkin and J. Matas, “All you need is a good init,” *4th Int. Conf. Learn. Represent. ICLR 2016 - Conf. Track Proc.*, Nov. 2015.

[20] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” 2010.

[21] C. Tan, F. Sun, T. Kong, W. Zhang, C. Yang, and C. Liu, “A survey on deep transfer learning,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018, vol. 11141 LNCS, pp. 270–279.

[22] J. Turian, L. Ratinov, Y. Bengio, and D. Roth, “A preliminary evaluation of word representations for named-entity recognition,” 2009.

[23] R. H. R. Hahnioser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, “Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit,” *Nature*, vol. 405, no. 6789, pp. 947–951, Jun. 2000.

[24] Y. Lecun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553. Nature Publishing Group, pp. 436–444, 27-May-2015.

[25] K. Abdelouahab, M. Pelcat, and F. Berry, “PhD Forum: Why TanH can be a Hardware Friendly Activation Function for CNNs,” 2017.

[26] H. Zhao, O. Gallo, I. Frosio, and J. Kautz, “Loss Functions for Image Restoration With Neural Networks,” *IEEE Trans. Comput. Imaging*, vol. 3, no. 1, pp. 47–57, Dec. 2016.

[27] C. M. Bishop, *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[28] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, Jun. 2017.

[29] D. E. Rumelhart and J. L. McClelland, “Learning Internal Representations by Error Propagation - MIT Press books,” in *Parallel Distributed Processing: Explorations in the Microstructure of Cognition: Foundations*, MIT Press, 1987, pp. 318–362.

[30] G. Hinton, N. Srivastava, and K. Swersky, “Neural Networks for Machine Learning Lecture 6a Overview of mini---batch gradient descent.”

[31] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.

[32] L. Cun *et al.*, “Handwritten Digit Recognition with a Back-Propagation Network,” 1990.

[33] J. Gu *et al.*, “Recent Advances in Convolutional Neural Networks,” Dec. 2015.

[34] J. J. Hopfield, “Neural networks and physical systems with emergent collective computational abilities.,” *Proc. Natl. Acad. Sci. U. S. A.*, vol. 79, no. 8, pp. 2554–2558, Apr. 1982.

[35] D. Kragic Jensfelt, “Tracking of Humans in Video Stream Using LSTM Recurrent Neural Net-work MASOUMEH POORMEHDI GHAEMMAGHAMI Master in Machine Learning,” 2017.

[36] J. Guo, “BackPropagation Through Time,” 2013.

[37] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[38] “Illustrated Guide to LSTM’s and GRU’s: A step by step explanation.” [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21. [Accessed: 27-Jun-2020].

[39] L. Leal-Taixé, A. Milan, I. Reid, S. Roth, and K. Schindler, "MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking," Apr. 2015.

[40] P. Dollar, R. Appel, S. Belongie, and P. Perona, "Fast feature pyramids for object detection," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, no. 8, pp. 1532–1545, Aug. 2014.

[41] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proceedings - 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR 2005*, 2005, vol. I, pp. 886–893.

[42] P. F. Felzenszwalb, R. B. Girshick, D. McAllester, and D. Ramanan, "Object detection with discriminatively trained part-based models," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 32, no. 9, pp. 1627–1645, 2010.

[43] F. Yang, W. Choi, and Y. Lin, "Exploit All the Layers: Fast and Accurate CNN Object Detector with Scale Dependent Pooling and Cascaded Rejection Classifiers," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-December, pp. 2129–2137.

[44] A. Geiger, P. Lenz, and R. Urtasun, "Are we ready for autonomous driving? the KITTI vision benchmark suite," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2012, pp. 3354–3361.

[45] "The KITTI Vision Benchmark Suite." [Online]. Available: http://www.cvlibs.net/datasets/kitti/. [Accessed: 22-Jun-2020].

[46] L. Wen *et al.*, "UA-DETRAC: A new benchmark and protocol for multi-object detection and tracking," *Comput. Vis. Image Underst.*, vol. 193, Apr. 2020.

[47] M. Andriluka, S. Roth, and B. Schiele, "People-tracking-by-detection and

people-detection-by-tracking," in *26th IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2008.

[48] J. Ferryman and A. Shahrokni, "PETS2009: Dataset and challenge," in *Proceedings of the 12th IEEE International Workshop on Performance Evaluation of Tracking and Surveillance, PETS-Winter 2009*, 2009.

[49] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (VOC) challenge," *Int. J. Comput. Vis.*, vol. 88, no. 2, pp. 303–338, Jun. 2010.

[50] T. Y. Lin *et al.*, "Microsoft COCO: Common objects in context," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2014, vol. 8693 LNCS, no. PART 5, pp. 740–755.

[51] "The PASCAL Visual Object Classes Challenge 2010 (VOC2010)." [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2010/index.html. [Accessed: 22-Jun-2020].

[52] "The PASCAL Visual Object Classes Challenge 2011 (VOC2011)." [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2011/index.html. [Accessed: 22-Jun-2020].

[53] "The PASCAL Visual Object Classes Challenge 2012 (VOC2012)." [Online]. Available: http://host.robots.ox.ac.uk/pascal/VOC/voc2012/index.html. [Accessed: 22-Jun-2020].

[54] "COCO - Common Objects in Context." [Online]. Available: http://cocodataset.org/#detection-eval. [Accessed: 10-Jun-2020].

[55] B. Wu and R. Nevatia, "Tracking of multiple, partially occluded humans

based on static body part detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2006, vol. 1, pp. 951–958.

[56] K. Bernardin and R. Stiefelhagen, "Evaluating Multiple Object Tracking Performance: The CLEAR MOT Metrics," *EURASIP J. Image Video Process. 2008 20081*, vol. 2008, no. 1, pp. 1–10, May 2008.

[57] E. Ristani, F. Solera, R. S. Zou, R. Cucchiara, and C. Tomasi, "Performance Measures and a Data Set for Multi-Target, Multi-Camera Tracking," Sep. 2016.

[58] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *Cvpr*, Jun. 2015.

[59] T. Uemura, H. Lu, and H. Kim, "Marine Organisms Tracking and Recognizing Using YOLO," Springer, Cham, 2020, pp. 53–58.

[60] S. J. Kim, J. Y. Nam, and B. C. Ko, "Online Tracker Optimization for Multi-Pedestrian Tracking Using a Moving Vehicle Camera," *IEEE Access*, vol. 6, pp. 48675–48687, Aug. 2018.

[61] G. Chandan, A. Jain, H. Jain, and Mohana, "Real Time Object Detection and Tracking Using Deep Learning and OpenCV," in *Proceedings of the International Conference on Inventive Research in Computing Applications, ICIRCA 2018*, 2018, pp. 1305–1308.

[62] W. Liu *et al.*, "SSD: Single Shot MultiBox Detector," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9905 LNCS, pp. 21–37, Dec. 2015.

[63] "COCO - Common Objects in Context." [Online]. Available: http://cocodataset.org/#home. [Accessed: 07-Jun-2020].

[64] R. Deepa, E. Tamilselvan, E. S. Abrar, and S. Sampath, "Comparison of Yolo, SSD, Faster RCNN for Real Time Tennis Ball Tracking for Action

Decision Networks," in *Proceedings of the 2019 International Conference on Advances in Computing and Communication Engineering, ICACCE 2019*, 2019.

[65] P. Nousi, I. Mademlis, I. Karakostas, A. Tefas, and I. Pitas, "Embedded UAV Real-Time Visual Object Detection and Tracking," 2020, pp. 708–713.

[66] D. Zhao, H. Fu, L. Xiao, T. Wu, and B. Dai, "Multi-Object Tracking with Correlation Filter for Autonomous Vehicle," *Sensors*, vol. 18, no. 7, p. 2004, Jun. 2018.

[67] "Main - Deep Systems / Artificial Intelligence Company." [Online]. Available: https://deepsystems.ai/. [Accessed: 07-Jun-2020].

[68] S. Baabou, A. G. Abubakr, F. Bremond, A. Ben Fradj, M. A. Farah, and A. Kachouri, "A Comparative Study and State-of-the-art Evaluation for Pedestrian Detection," in *19th International Conference on Sciences and Techniques of Automatic Control and Computer Engineering, STA 2019*, 2019, pp. 485–490.

[69] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, "Simple Online and Realtime Tracking," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-August, pp. 3464–3468, Feb. 2016.

[70] F. Yu, W. Li, Q. Li, Y. Liu, X. Shi, and J. Yan, "POI: Multiple Object Tracking with High Performance Detection and Appearance Feature," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9914 LNCS, pp. 36–42, Oct. 2016.

[71] S. Bell, C. L. Zitnick, K. Bala, and R. Girshick, "Inside-Outside Net: Detecting Objects in Context with Skip Pooling and Recurrent Neural Networks," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 2874–2883, Dec. 2015.

[72] S. Gidaris and N. Komodakis, "Object detection via a multi-region and

semantic segmentation-aware U model," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[73] W. Min, M. Fan, X. Guo, and Q. Han, "A New Approach to Track Multiple Vehicles with the Combination of Robust Detection and Two Classifiers," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 1, pp. 174–186, Jan. 2018.

[74] P. Bergmann, T. Meinhardt, and L. Leal-Taixe, "Tracking without bells and whistles," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-October, pp. 941–951, Mar. 2019.

[75] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, "Multiple hypothesis tracking revisited," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[76] N. Wojke, A. Bewley, and D. Paulus, "Simple Online and Realtime Tracking with a Deep Association Metric," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2017-September, pp. 3645–3649, Mar. 2017.

[77] E. Ristani and C. Tomasi, "Features for Multi-Target Multi-Camera Tracking and Re-Identification," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 6036–6046, Mar. 2018.

[78] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-December, pp. 770–778.

[79] "ImageNet." [Online]. Available: http://www.image-net.org/. [Accessed: 20-Jun-2020].

[80] H.-L. Ooi, G.-A. Bilodeau, and N. Saunier, "Supervised and Unsupervised Detections for Multiple Object Tracking in Traffic Scenes: A Comparative Study," Mar. 2020.

[81] K. Zhou, Y. Yang, A. Cavallaro, and T. Xiang, "Omni-Scale Feature

Learning for Person Re-Identification," May 2019.

[82] T. Matsukawa and E. Suzuki, "Person re-identification using CNN features learned from combination of attributes," in *Proceedings - International Conference on Pattern Recognition*, 2016, vol. 0, pp. 2428–2433.

[83] S. Karthik, A. Prabhu, and V. Gandhi, "Simple Unsupervised Multi-Object Tracking," Jun. 2020.

[84] R. Pflugfelder, "An In-Depth Analysis of Visual Tracking with Siamese Neural Networks," Jul. 2017.

[85] L. Bertinetto, J. Valmadre, J. F. Henriques, A. Vedaldi, and P. H. S. Torr, "Fully-Convolutional Siamese Networks for Object Tracking," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9914 LNCS, pp. 850–865, Jun. 2016.

[86] R. Tao, E. Gavves, and A. W. M. Smeulders, "Siamese Instance Search for Tracking," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-December, pp. 1420–1429, May 2016.

[87] J. Valmadre, L. Bertinetto, J. Henriques, A. Vedaldi, and P. H. S. Torr, "End-to-end representation learning for Correlation Filter based tracking," 2017.

[88] B. Li, W. Wu, Q. Wang, F. Zhang, J. Xing, and J. Yan, "SiamRPN++: Evolution of Siamese Visual Tracking with Very Deep Networks," Dec. 2018.

[89] B. Shuai, A. G. Berneshawi, D. Modolo, and J. Tighe, "Multi-Object Tracking with Siamese Track-RCNN," Apr. 2020.

[90] "SiameseFC tracker." [Online]. Available: https://www.robots.ox.ac.uk/~luca/siamese-fc.html. [Accessed: 22-Jun-2020].

[91] "The Seventh Visual Object Tracking VOT2019 Challenge Results :: ViCoS Prints." [Online]. Available: http://prints.vicos.si/publications/375. [Accessed: 22-Jun-2020].

[92] "The sixth Visual Object Tracking VOT2018 challenge results :: ViCoS Prints." [Online]. Available: http://prints.vicos.si/publications/365. [Accessed: 22-Jun-2020].

[93] M. Kristan *et al.*, "The Visual Object Tracking VOT2017 Challenge Results," in *Proceedings - 2017 IEEE International Conference on Computer Vision Workshops, ICCVW 2017*, 2017, vol. 2018-January, pp. 1949–1972.

[94] L. Leal-Taixé, C. C. Ferrer, and K. Schindler, "Learning by tracking: Siamese CNN for robust target association," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, pp. 418–425, Apr. 2016.

[95] M. Kim, S. Alletto, and L. Rigazio, "Similarity Mapping with Enhanced Siamese Network for Multi-Object Tracking," Sep. 2016.

[96] C. Kim, F. Li, and J. M. Rehg, "Multi-object Tracking with Neural Gating Using Bilinear LSTM," 2018.

[97] L. Lan, X. Wang, G. Hua, T. S. Huang, and D. Tao, "Semi-online Multi-people Tracking by Re-identification," *Int. J. Comput. Vis.*, pp. 1–19, Mar. 2020.

[98] S. Liao, Y. Hu, X. Zhu, and S. Z. Li, "Person re-identification by Local Maximal Occurrence representation and metric learning," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2015, vol. 07-12-June-2015, pp. 2197–2206.

[99] A. Milan, S. H. Rezatofighi, A. Dick, I. Reid, and K. Schindler, "Online Multi-Target Tracking Using Recurrent Neural Networks," Apr. 2016.

[100] A. Sadeghian, A. Alahi, and S. Savarese, "Tracking The Untrackable:

Learning To Track Multiple Cues with Long-Term Dependencies," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2017-October, pp. 300–311, Jan. 2017.

[101] T. Wang *et al.*, "Spatio-Temporal Point Process for Multiple Object Tracking," *IEEE Trans. Neural Networks Learn. Syst.*, pp. 1–12, Jun. 2020.

[102] P. Dendorfer *et al.*, "MOT20: A benchmark for multi object tracking in crowded scenes," Mar. 2020.

[103] J. Xiang, G. Zhang, J. Hou, N. Sang, and R. Huang, "Multiple Target Tracking by Learning Feature Representation and Distance Metric Jointly," Feb. 2018.

[104] R. Henschel, L. Leal-Taixé, D. Cremers, and B. Rosenhahn, "Fusion of Head and Full-Body Detectors for Multi-Object Tracking," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 1509–1518, May 2017.

[105] Z. Lian, S. Shao, and C. Huang, "A Real Time Face Tracking System based on Multiple Information Fusion," *Multimed. Tools Appl.*, pp. 1–19, May 2020.

[106] Y. Zhang *et al.*, "Long-term Tracking with Deep Tracklet Association," *IEEE Trans. Image Process.*, pp. 1–1, May 2020.

[107] A. Hermans, L. Beyer, and B. Leibe, "In Defense of the Triplet Loss for Person Re-Identification," Mar. 2017.

[108] Y. Xu, A. sep, Y. Ban, R. Horaud, L. Leal-Taixe, and X. Alameda-Pineda, "How to Train Your Deep Multi-Object Tracker," 2020.

[109] W. Feng, Z. Hu, W. Wu, J. Yan, and W. Ouyang, "Multi-Object Tracking with Multiple Cues and Switcher-Aware Classification," Jan. 2019.

[110] P. Chu and H. Ling, "FAMNet: Joint Learning of Feature, Affinity and Multi-dimensional Assignment for Online Multiple Object Tracking," *Proc.*

IEEE Int. Conf. Comput. Vis., vol. 2019-October, pp. 6171–6180, Apr. 2019.

[111] K. Yoon, J. Gwak, Y. M. Song, Y. C. Yoon, and M. G. Jeon, "OneShotDA: Online Multi-Object Tracker with One-Shot-Learning-Based Data Association," *IEEE Access*, vol. 8, pp. 38060–38072, 2020.

[112] O. Vinyals, C. Blundell, T. Lillicrap, K. Kavukcuoglu, and D. Wierstra, "Matching NetworksV2," in *Advances in neural information processing systems*, 2016, pp. 3630–3638.

[113] J. Xu, Y. Cao, Z. Zhang, and H. Hu, "Spatial-Temporal Relation Networks for Multi-Object Tracking," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2019-October, pp. 3987–3997, Apr. 2019.

[114] L. Chen, H. Ai, Z. Zhuang, and C. Shang, "Real-time Multiple People Tracking with Deeply Learned Candidate Selection and Person Re-Identification," *Proc. - IEEE Int. Conf. Multimed. Expo*, vol. 2018-July, Sep. 2018.

[115] Y.-C. Yoon, D. Y. Kim, K. Yoon, Y. Song, and M. Jeon, "Online Multiple Pedestrian Tracking using Deep Temporal Appearance Matching Association," Jul. 2019.

[116] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[117] G. D. Evangelidis and E. Z. Psarakis, "Parametric image alignment using enhanced correlation coefficient maximization," *IEEE Trans. Pattern Anal. Mach. Intell.*, 2008.

[118] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, "YOLOv4: Optimal Speed and Accuracy of Object Detection," Apr. 2020.

[119] D. Misra, "Mish: A Self Regularized Non-Monotonic Neural Activation

Function," Aug. 2019.

[120] "GitHub - AlexeyAB/darknet: YOLOv4 - Neural Networks for Object Detection (Windows and Linux version of Darknet)." [Online]. Available: https://github.com/AlexeyAB/darknet. [Accessed: 22-Jul-2020].

[121] "Darknet: Open Source Neural Networks in C." [Online]. Available: https://pjreddie.com/darknet/. [Accessed: 22-Jul-2020].

[122] angela Casado-García and Jónathan Heras, "Ensemble Methods for Object Detection," 2020.

[123] C. Eggert, S. Brehm, A. Winschel, D. Zecha, and R. Lienhart, "A closer look: Small object detection in faster R-CNN," in *Proceedings - IEEE International Conference on Multimedia and Expo*, 2017, pp. 421–426.

[124] S. Romberg, L. G. Pueyo, R. Lienhart, and R. Van Zwol, "Scalable logo recognition in real-world images," in *Proceedings of the 1st ACM International Conference on Multimedia Retrieval, ICMR'11*, 2011, pp. 1–8.

[125] F. Schroff, D. Kalenichenko, and J. Philbin, "FaceNet: A Unified Embedding for Face Recognition and Clustering," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 07-12-June-2015, pp. 815–823, Mar. 2015.

[126] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W. Wong, and W. Woo, "Convolutional LSTM Network: A Machine Learning Approach for Precipitation Nowcasting," *Adv. Neural Inf. Process. Syst.*, vol. 2015-January, pp. 802–810, Jun. 2015.

[127] T. Siddiqui and S. Bharadwaj, "Future semantic segmentation of time-lapsed videos with large temporal displacement," Dec. 2018.

[128] F. A. Elshwemy, R. Elbasiony, and M. T. Saidahmed, "A New Approach for Thermal Vision based Fall Detection Using Residual Autoencoder," *Int. J. Intell. Eng. Syst.*, vol. 13, no. 2, 2020.

[129] S. Ioffe and C. Szegedy, “Batch normalization: Accelerating deep network training by reducing internal covariate shift,” in *32nd International Conference on Machine Learning, ICML 2015*, 2015, vol. 1, pp. 448–456.

[130] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna, “Rethinking the Inception Architecture for Computer Vision,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-December, pp. 2818–2826.

[131] G. E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. R. Salakhutdinov, “Improving neural networks by preventing co-adaptation of feature detectors,” Jul. 2012.

[132] J. Hermans, G. Spanakis, and R. Möckel, “Accumulated Gradient Normalization,” *J. Mach. Learn. Res.*, vol. 77, pp. 439–454, Oct. 2017.