REPUBLIQUE ALGERIENNE DEMOCRATIQUE ET POPULAIRE
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mohamed Khider – BISKRA
Faculté des Sciences Exactes, des Sciences de la Nature et de la Vie
Département d'informatique

N° d'ordre : IVA11/M2/2021

Mémoire

Présenté pour obtenir le diplôme de master académique en **Informatique**

Parcours: **Image et Vie Artificielle (IVA)**

Real Time Multi-Object Tracking Using Deep Learning

Par:
ABDELAZIZ HAMADI

Soutenu le ../07/2021 devant le jury composé de :

Nom Prénom	grade	Président
Zerari Abd El Mouméne	MCB	Rapporteur
Nom Prénom	grade	Examinateur

Année universitaire 2020-2021

Dedication

This work is dedicated to

My beloved Parents who always supported and believed in me.

My brother and sister Tahar and Youssra for their encouragements.

Last but not least, the boys.

And to all those who I love and cherich.

Acknowledgement:

First and foremost, I would like to show gratitude and praise to god almighty for giving me the strength, knowledge, ability and opportunity to undertake this challenge and to persevere.
and complete it satisfactorily.

I wish to express my sincere appreciation to Dr.Zerari for his valuable guidance.

I also like to thank the jury members for their efforts to evaluate this work.

And finally, I wish to express my deepest gratitude to to my classmates and the academic crew of the department for their assistance provided through my academic career.

Abstract

The Deep Learning techniques has proven its effectiveness in so many computer vision (CV) tasks and one of those are the multi object detection and multi object tracking. Which comes in a lot of shapes and forms in terms of precision and speed.

This master thesis goal falls under the Online Tracking detection-based method, to be able to manage Real-time Live scenarios where precision and speed are required for a quick and accurate response if needed. Where in our work, we only count on the current frames and past information (No peaking to future frames) to predict trajectories and track objects, which in our case is the “Pedestrian class”, we will use state-of-the-art and novel CNN based techniques in the object detection phase and in object tracking phase.

We propose a better approach for an online Pedestrian tracking algorithm that is a robust and fast for an online application use, by reimplementing state-of-the-art algorithm DeepSORT, but instead of using its default detector Faster R-CNN, we replace it with a better state-of-the-art multi-object detector called YOLOv4.

Since YOLOv4 didn’t give satisfactory results in terms of pedestrian detecting class specially in situations where there’s low light, extremely low light condition, or greyscale inputs, which means less information to process. And these kinds of situation are very important in terms of Pedestrian tracking and for security measurement, that’s duo to the fact that YOLOv4 was designed to be a good multi-class object detector where its goal to localize different object classes (80 class) as much as possible from a given scene.

So, we customize the Object detector for our needs which we want a high precision in Pedestrian localization specially in low light and extremely low light conditions, with keeping the speed factor as important as the Precision, by customizing the architecture to focus on the localization of pedestrian’s class only, and retrain the model with a custom Pedestrian specified dataset. With the transfer learning techniques.

After the preparation phase we reimplement the model with the new obtained weights to a TensorFlow format so it can be used in a GPU environment for faster object detecting calculation, to feed DeepSORT algorithm with good detections as much and as fast as possible.

Then we build a bridge between the outputs of our model “Pedestrian_YOLOv4” and the inputs of DeepSORT algorithm so it can perform data association between the outputted detections in every frame by our model.

Our experimental evaluation demonstrates that our techniques can produce high number of tracks with good overall precision, and an improvement in terms of speed specially in greyscale inputs and extremely low light conditions.

Résumé

Les techniques d'apprentissage en profondeur ont prouvé leur efficacité dans de nombreuses tâches de vision par ordinateur (CV) et l'une d'entre elles est la détection et le suivi multi-objets. Ce qui se présente sous de nombreuses formes et formes en termes de précision et de vitesse. Ce mémoire de master s'inscrit dans la méthode basée sur la détection du suivi en ligne, pour pouvoir gérer des scénarios en temps réel en direct où la précision et la vitesse sont requises pour une réponse rapide et précise si nécessaire. Dans notre travail, nous comptons uniquement sur les images actuelles et les informations passées pour prédire les trajectoires et suivre les objets, qui dans notre cas est la "classe piétonne", nous utiliserons de nouvelles techniques basées sur CNN dans la phase de détection d'objet et dans la phase de suivi d'objet.

Nous proposons une meilleure approche pour un algorithme de suivi des piétons en ligne qui soit robuste et rapide pour une utilisation d'une application en ligne, en réimplémentant l'algorithme de pointe DeepSORT, mais au lieu d'utiliser son détecteur par défaut Faster R-CNN, nous le remplaçons par un meilleur détecteur multi-objets de pointe appelé YOLOv4.

Étant donné que YOLOv4 n'a pas donné de résultats satisfaisants en termes de classe de détection des piétons, en particulier dans les situations où il y a une faible luminosité, des conditions de luminosité extrêmement faibles ou des entrées en niveaux de gris, ce qui signifie moins d'informations à traiter. Et ce genre de situation est très important en termes de pistage des piétons et pour la mesure de sécurité, c'est dû au fait que YOLOv4 a été conçu pour être un bon détecteur d'objets multi-classes où son objectif de localiser différentes classes d'objets (classe 80) autant que possible à partir d'une scène donnée.

Ainsi, nous personnalisons le détecteur d'objets pour nos besoins que nous voulons une haute précision dans la localisation des piétons, en particulier dans des conditions de faible luminosité et de luminosité extrêmement faible, tout en gardant le facteur de vitesse aussi important que la précision, en personnalisant l'architecture pour se concentrer sur la localisation de classe de piétons uniquement, et recyclez le modèle avec un jeu de données personnalisé spécifié pour les piétons. Avec les techniques d'apprentissage par transfert.

Après la phase de préparation, nous réimplémentons le modèle avec les nouveaux poids obtenus au format TensorFlow afin qu'il puisse être utilisé dans un environnement GPU pour un calcul de détection d'objet plus rapide, pour alimenter l'algorithme DeepSORT avec de bonnes détections autant et aussi rapidement que possible.

Ensuite, nous construisons un pont entre les sorties de notre modèle "Pedestrian_YOLOv4" et les entrées de l'algorithme DeepSORT afin qu'il puisse effectuer une association de données entre les détections sorties dans chaque trame par notre modèle.

Notre évaluation expérimentale démontre que nos techniques peuvent produire un nombre élevé de pistes avec une bonne précision globale et une amélioration en termes de vitesse, en particulier dans les entrées en niveaux de gris et dans des conditions d'éclairage extrêmement faibles.

List of Figures

List of Tables

List of content

General Introduction

The goal of this research is to apply deep learning and tracking-by-detection approaches to solve the challenge of visual tracking of many objects. The context of this project is explained in this chapter.

Multiple Object Tracking in Computer Vision

The Multiple Object Tracking or MOT is a vital computer vision issue which continues to pull in consideration since of its potential in both the academic and commercial circles. The real-world applications of the multiple object tracking are numerous including human-computer interaction, independent vehicles, mechanical technology, video indexing, surveillance or security, among others. The computer vision community have been making huge endeavours within the past few decades to illuminate the MOT issue but the assignment is still open for improvement.

Numerous autonomous car projects are taking put all inclusive which require arrangements to various different issues counting to keep an eye to all other moving objects within the region where the car is found (Figure 1.1). The outputs from the following module are a fundamental input for other modules like move planning and direction planning. Autonomous vehicles are key within the persistent advance made in tracking and within the computer vision community in general. Many multi-object tracking calculations have been proposed to unravel the problem of real-world activity monitoring. In these kinds of assignments, the calculations ought to deal with complex occlusion circumstances and troublesome object matching.

moreover, it is being utilized for tracking the hand movements with this following ready to create exceptionally curious applications that range from anticipating sign language to recreations like playing “hand ping pong”.

One of the foremost considered tracking zones is the person on foot tracking, basically since this particular kind of object can be seen in a expansive number of applications with commercial potential. As a few studies show [1], approximately the 70% of the current investigate done in MOT is committed to people on foot. The trouble of MOT lies in different challenging situations that can happen such as variety of the illumination, variety of scale, target deformation or quick movement. Most of these challenges are common to Single Object Tracking (SOT) but MOT too has to unravel two primary tasks: deciding the number of objects and maintaining its identities over the time. moreover it is being utilized for tracking the hand movements With this following ready to create exceptionally curiously applications that range from anticipating sign language to recreations like playing “hand ping pong”. One of the foremost considered tracking zones is the person on foot tracking, basically since this particular kind of object can be seen in a expansive number of applications with commercial potential. The trouble of MOT lies in different challenging situations that can happen such as variety of the illumination, variety of scale, target deformation or quick movement. Most of this challenges are common to Single Object Tracking (SOT) but MOT too has to unravel two primary tasks: deciding the number of objects and maintaining its identities over the time.

Deep Learning in Object Detection

Given the broad areas of application where deep learning is succeeding and the ones who are still in research it is going to be studied on this master thesis the use of deep learning techniques to tackle the

multi-object tracking problem. The deep learning for tracking has been used in previous works from others such as CenterTrack (by Xingyi Zhou, Vladlen Koltun and Philipp Krahenbuhl) [27]. In his work, the detections obtained by the neural networks allow to build a hybrid tracker also known as tracking-by-detection. However, it localizes objects and predicts their associations with the previous frame. Another work named DEFT, or "Detection Embeddings for Tracking." (By Mohamed Chaabane, Peter Zhang, J. Ross Beveridge, Stephen O'Hara) [3], relies on an appearance-based object matching network jointly-learned with an underlying object detection network. Other interesting works with neural networks, in this case, for object detection have been developed such as YOLO family [4]. A new approach to object detection is called You Only Look Once (YOLO) which means that an image can be predicted what the objects are and where they are at one glance. modules needed are: a Camera that provides the images (such as a webcam, a video or via remote proxy) and a Detection Network that encapsulates an object detector neural network. As a result, this node allows the user to visualize object detections, i.e. bounding boxes drawn over the image in real-time. It also provides functionality to perform on-demand detection. The Figure 1.5 shows the Object Detector running.

Example of the Object Detector node working in real-time (yolov4)

Objective

The main objective of this master thesis is to construct a multi-Pedestrian tracking application with a multi-object method, which makes utilize of two methods: deep learning and 2D-tracking. In this work we are aiming to study how to utilize the finest of both techniques to construct a strong and fast multi-Pedestrian tracker which can be able of run-in asset constrained equipment on real-time. This work takes the form of a deep study on the domain to know which are the best in both words in terms of tracking and in terms of detecting.

Experiment multiple object detector algorithms using deep learning

Learn the essentials of object detection using deep learning techniques. Study the performance on both accuracy and speed of these techniques with datasets and benchmarks. Then Choosing the best fit algorithm Finally, select the default object detector.

1. Development of object tracker using Deep Learning

Learn the basics of object tracking. Methods to extract the important features of each object fed by the selected object detector, to create a reliable object tracker and assign IDs to each object detected with considering accuracy and speed.

2. Combining neural object detector & tracker in one software component

Creating an application that provides synchronization between them and make the results visible for the user (bounding boxes with their label on each object detected and even in saved file for later analysis use).

3. GPU Implementation

Re-implementing the selected object detector with TensorFlow-GPU for faster calculation during application time.

4. Experimental validation

Finally, several experiments with real hard conditions as Grayscale Inputs (less information), and extremely low light conditions to test its robustness in various situations, and comparing FPS results will be performed to validate the developed solution and extract results.

Chapter 1: Object Tracking

Introduction

The main focus in object tracking is to know the state of the target (object) over a sequence of images (frames). This state can be known by different characteristic such as form (structure), appearance, location or speed. It is a hard field of study since many difficulties must be solved by various algorithms. Among them the management of variations in lighting and the perspective of the object (camera rotation) that can lead to changes in the appearance of an object. Likewise, the occlusions that occur when objects are mixed with other object or component of the scene or the quality of the image itself may be a problem.

1.1– Different Tracking Algorithms

1.1.1- Tracking Using Matching

These algorithms make comparison between the representation of the model of an object created from frame T-1 and the possible candidates in the frame T, these methods rely on similarity measurement. Some of the most powerful methods are:

- **Mean shift tracking [36]:** in the sequel the support of two modules which should provide (a) detection and localization in the initial frame of the objects to track (targets) and (b) periodic analysis of each object to account for possible updates of the target models due Signiant changes in colour.
- **Kalman appearance tracker [35]:** this algorithm faces the occlusion handling. The tracking process is performed using an appearance-based tracking algorithm, in which Kalman filtering is prepared to bring in particle filter to solve the heavy occlusion problems.

- **Lucas-Kanade tracker [37]:** Since the Lucas-Kanade algorithm was proposed in 1981 image alignment has become one of the most widely used techniques in computer vision, the Lucas–Kanade method is a widely used differential method for optical flow estimation developed by Bruce D. Lucas and Takeo Kanade. It assumes that the flow is essentially constant in a local neighbourhood of the pixel under consideration, and solves the basic optical flow (Optical flow or optic flow is the pattern of apparent motion of objects, surfaces, and edges in a visual scene caused by the relative motion between an observer and a scene) equations for all the pixels in that neighbourhood.

1.1.2 – Tracking by detection:

A model is built to separate an object from the background [38]. Once you have one detection it is associated with the previous detections. Currently, the studies are heading towards neural networks to extract detections. Which is this master thesis going to be focusing on.

Prior to the modern techniques to be discussed here there are more “classic ways” of tracking objects that can be useful in problems that require real time, for example. One of the most well-known is feature tracking. This technique uses characteristic points that can be found in images and that allow to estimate the movement. These points must meet some requirements to be able to be characteristic of the image such as repeatability (the characteristic can be found in the images even if they have

undergone some transformation), compatibility (each characteristic must be descriptive and easy to find) or efficiency (the representation of the information characteristic of the image must be done with as few characteristics as possible). The characteristic points most commonly used are corners. They are characterized by gradients with higher values in them in two or more directions. These techniques can be seen in Harris [92] and Shi-Tomasi corner detectors [93].

There are tracking systems that take advantage of the speed of feature tracking and the accuracy of neural networks to create a "hybrid tracking". In this type of tracking the detections are done each N frames using some type of neural network and the intermediate tracking is done through feature tracking. With the arrival of neural networks this way of grouping the tracking methods changes to adapt to them [94]:

- **Tracking-by-detection**: They are designed to follow a certain class of object (model-based) and to obtain a specific classifier. In practice, the detections are obtained with neural networks and they are linked in tracking using temporal information. They are limited to a single class of objects.
- **Tracking, learning and detection:** They are characterized by being fully trained online. A typical tracker example of this group samples zones closes to the object and considers them foreground, the same happens with the distant zones that would be assigned to the background. With this a classifier can be built that differentiates them and estimates the new location of the object in the following frame [95]. It has been tried to introduce neural networks in environments with online training but due to the slowness of the networks when training the results are slow in practice.
- **Siamese-based tracking:** Multiple patch candidates from the new frame are received and the one with the highest matching score with respect to the previous frame is chosen as the best candidate, that is, the most similar according to the matching function.
- **Tracking as regression:** In this group, on the other hand, the network receives only two images (the previous frame and the current one) and directly returns the location of the object in the current frame. Since this tracker predicts a bounding box instead of just the position, it is able to model changes in scale and aspect of the tracked template. However, it only can process a single target and it needs from data augmentation techniques to learn all possible transformations of the targets [94].

1.2 – Brief Background on DEEP LEARNING (DL)

Deep leaning is still a term that has shadows in its definition, for many changes that it took throughout the years, a modern definition can be found in GOODfellow's book "Deep Learning" [5], it is a tool that allows computer to lean from experience.

1.2.1- Machine learning overview (ML)

As mentioned in the introduction chapter, Machine Learning is a subfield of Artificial Intelligence (AI) [2], ML models use statistical tools to estimate human behaviour, since it is nearly impossible to write algorithms for human behaviours like recognition of hand-written numbers duo to a lot of variation in the user inputs, and it will be very handy to use ML, because it uses statistical models to build knowledge about the dataset, (figure 1.1) shows an application where ML is utilized as a classifier for hand-written numbers.

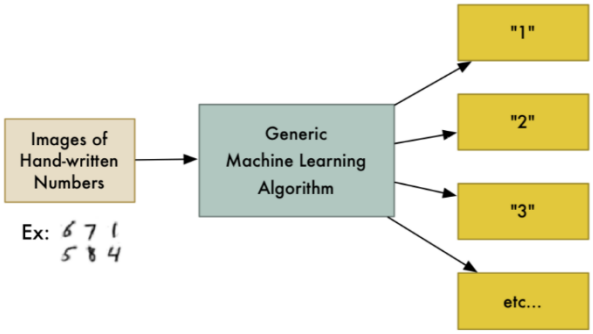


Figure 1.1: Classifying hand-written numbers with machine learning [13].

Machine learning can be split in three main approaches:

1.2.1.1 – Supervised Learning

In supervised learning, we use labelled 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 (figure 1.2), 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 Deep Learning DL object detection models are trained with this approach.

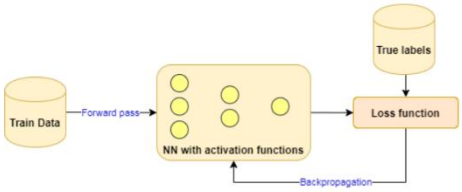


Figure 1.2. Supervised learning based NN pipeline

The most performing tasks with supervised learning are:

- **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.

- **Regression:** 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. Like Predicting the coordinates of digits in a license plate is a regression task.

1.2.1.2 – Unsupervised Learning:

Unsupervised learning the reverse of supervised learning. The goal here is to find structure inside the given input which corresponds to the problem to be predicted but starting from the non-labelled data, the absence of annotated data or learning base which means unsupervised learning.

1.2.1.3 - 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.

1.2.2 – Artificial Neural Network (ANN)

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 [7]. 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 perceptron's (basic units) in ANN. As depicted in Figure 1.3, the perceptron can take several inputs $\{x1, x2, \ldots, xn\}$ that might represent pixels in an image. These inputs are multiplied with the associated weight (wn in the n th connection). The weighted sum is passed to an activation function φ (.), 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 (1.1). $\boldsymbol{w}.\boldsymbol{x}$ represent dot product between the weights and inputs vectors.

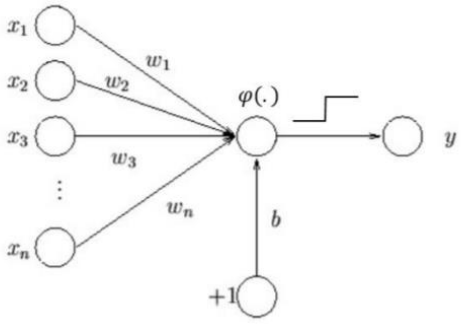


Figure 1.3 A perceptron with n inputs [15]

$$y(\boldsymbol{x}) = \begin{cases} 1, & if\ \varphi(\boldsymbol{w}.\boldsymbol{x} + b) \geq\ threshold \\ -1, & otherwise \end{cases} \quad (1.1)$$

Multilayer Neural Network is capable of solving nonlinear mapping such as the XOR problem (The XOR, or "exclusive or", problem is a classic problem in ANN research. It is the problem of using a neural network to predict the outputs of XOR logic gates given two binary inputs. An XOR function should return a true value if the two inputs are not equal and a false value if they are equal). Usually, a NN with more than two layers is referred to as deep NN, a classifier must be able to learn by giving examples. It starts by initializing weights and after that we pass example through the network, the error the models make is calculated with a loss function and fed backwards to the network through a process called "backpropagation" to update the weighs.

1.2.3 - Convolutional Neural Network (CNN)

Among different types of deep neural networks, Convolutional Neural Networks (CNN) has been studied extensively. CNN is the same as ANN but with at least one convolutional layer (CN). In 1990, LeCun et al. [9] 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. [10] published their architecture AlexNet, 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.

- **Convolution layer:** The goal of convolutional layer is 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 1.4, 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.

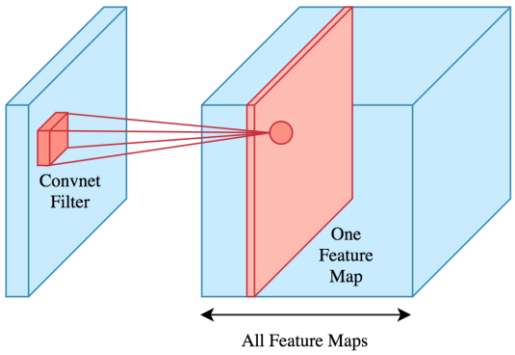


Figure 1.4. convolution kernel operating on the input image to output feature map.

The number of feature maps is the same number of kernels, and we can determine the feature map shape by this equation:

$$f(n) = \frac{i(n)-k(n)+2p(n)}{s(n)} + 1 \, , where \; n = 1,2 \dots N \qquad (1.2)$$

Where:

f(n): the feature map shape in the axis n

i, k: Input shape and kernel shape, in the axis n, respectively.

p: The amount zero-padding to add on the axis n.

s: The amount of stride length by which the filter shifts.

- **pooling layer:** The pooling layer goal is to achieve shift-invariance by providing an approach to down-sampling feature maps. usually placed between two convolutional layers. Figure 1.5 shows two standard pooling methods; average pooling and max pooling. Note that not the same as convolution kernels, the pooling layer has no learning parameters. We can determine the shape the same way as feature maps.

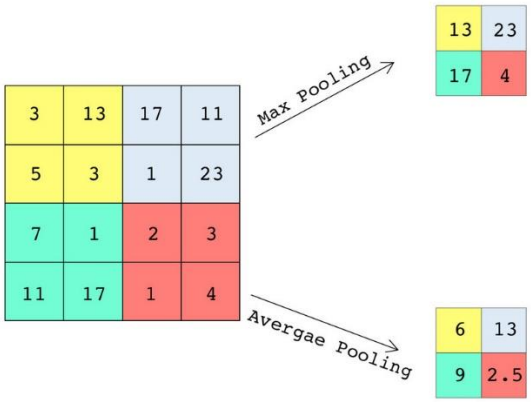


Figure 1.5. example of max-pooling and average-pooling.

- **Fully-connected layer (FC):** Fully-connected 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 1.6 shows the first CNN framework published by LeCun et al. [11]. 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.

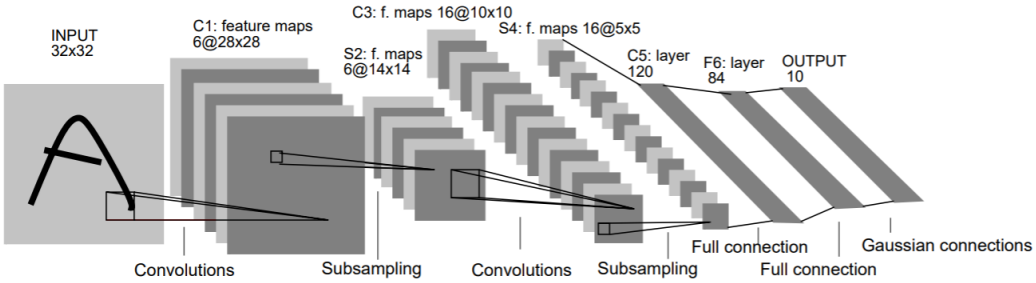


Figure 1.6. LeNet-5 network [11].

1.3 – Multi-Object tracking (MOT): metrics and datasets

Besides speaking about datasets and MOT algorithms that are used to solve this problem, it is necessary to use a set of measurements that provides an evaluation of the performance of a given solution. Since in this master thesis we are going to be focusing on *tracking-by-detection* technique, we are going to see both the detection metrics and tracking metrics. But first let's talk about datasets.

1.3.1 – Datasets

1.3.1.1 – Object detection datasets

Detecting an object has to state that an object belongs to a specified class and locate it in the image. The localization of an object is mostly represented by a bounding box as shown in Fig. 3.1.1. Using challenging datasets as benchmark is significant in many areas of research, because they are able to draw a standard comparison between different algorithms and set goals for solutions. Another challenge is the detection of pedestrians for which several datasets have been created. The Caltech Pedestrian Dataset [19] contains 350,000 labelled instances with bounding boxes. General object detection datasets like PASCAL VOC [20], MS COCO [21], ImageNet-loc [22] are the mainstream benchmarks of object detection task. The official metrics are mainly adopted to measure the performance of detectors with corresponding dataset. We will site few among the high-quality datasets.

- **PASCAL VOC Dataset:** for the detection of basic object categories, a multi-year effort from 2005 to 2012 was devoted to the creation and maintenance of a series of benchmark datasets that were widely adopted. The PASCAL VOC datasets [20] contain 20 object categories (in VOC2007, such as person, bicycle, bird, bottle, dog, etc.) spread over 11,000 images. The 20 categories can be considered as 4 main branches-vehicles, animals, household objects and people. Some of them increase semantic specificity of the output, such as car and motorbike, different types of vehicles, but not look similar. In addition, the visually similar classes increase the difficulty of detection, e.g., dog vs. cat. Over 27,000 object instance bounding boxes are labelled, of which almost 7,000 have detailed segmentations. Imbalanced datasets exist in the VOC2007 dataset, while the class person is definitely the biggest one, which is approximately 20 times more than the smallest class sheep in the training set. This problem is widespread in the surrounding scene and how can detectors solve this well? Another issue is viewpoint, such as, front, rear, left, right and unspecified, the detectors need to treat different viewpoints separately.

- **MS COCO benchmark:** The Microsoft Common Objects in Context (MS COCO) dataset [21] for detecting and segmenting objects found in everyday life in their natural environments contains 91 common object categories with 82 of them having more than 5,000 labelled instances. These categories cover the 20 categories in PASCAL VOC dataset. In total the dataset has 2,500,000 labelled instances in 328,000 images. MS COCO dataset also pays attention to varied viewpoints and all objects of it are in natural environments which gives us rich contextual information. In contrast to the popular ImageNet dataset [22], COCO has fewer categories but more instances per category. The dataset is also significantly larger in the number of instances per category (27k on average) than the PASCAL VOC datasets [20] (approximately 10 times less than MS COCO dataset) category. As we know, small objects need more contextual reasoning to recognize. Images among MS COCO dataset are rich in contextual information. The biggest class is also the person, nearly 800,000 instances in the whole dataset. Another small class is hair brush whose number is nearly 800.
- **ImageNet benchmark:** Challenging datasets can encourage a step forward of vision tasks and practical applications. Another important large-scale benchmark dataset is ImageNet dataset [22]. The ILSVRC task of object detection evaluates the ability of an algorithm to name and localize all

instances of all target objects present in an image. ILSVRC2014 has 200 object classes and nearly 450k training images, 20k validation images and 40k test images.

- **Open Image v5 /v6:** Open Images [23] is a dataset of 9.2M images annotated with image-level labels, object bounding boxes, object segmentation masks, and visual relationships. Open Images V5 contains a total of 16M bounding boxes for 600 object classes on 1.9M images, which makes it the largest existing dataset with object location annotations. First, the boxes in this dataset have been largely manually drawn by professional annotators (Google-internal annotators) to ensure accuracy and consistency. Second, the images in it are very diverse and mostly contain complex scenes with several objects (8.3 per image on average). Third, this dataset offers visual relationship annotations, indicating pairs of objects in particular relations (e.g.," woman playing guitar"," beer on table"). In total it has 329 relationship triplets with 391,073 samples. Fourth, V5 provides segmentation masks for 2.8M object instances in 350 classes. Segmentation masks mark the outline of objects, which characterizes their spatial extent to a much higher level of detail. Finally, the dataset is annotated with 36.5M image level labels spanning 19,969 classes.

1.3.1.2 – Object tracking datasets

In this section we represent the most vital datasets, MOT dataset that targets specifically pedestrians tracking, and KITTI datasets targets vehicle tracking. They are both characterized by crowd sequences, different viewpoints, various illumination levels, and camera motions.

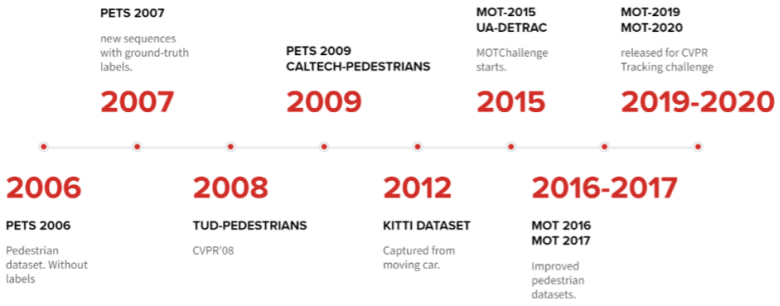


Figure 1.7 Timeline for the most used MOT datasets.

- **MOTChallenge [24]:** MOTChallenge is the most commonly used benchmark for multiple object tracking. It provides, among others, some of largest datasets for pedestrian tracking that are currently publicly available. For each dataset, the ground truth for the training split, and detections for both training and test splits are provided. The reason why MOTChallenge datasets frequently provide detections (often referred to as public detections, as opposed to the private detections, that are obtained by the algorithm authors by using a detector of their own) is that the detection quality has a big impact on the final performance of the tracker, but the detection part of the algorithms is often independent from the tracking part and usually uses already existing models; providing public detections that every model can use makes the comparison of the tracking algorithms easier, since the detection quality is factored out from the performance computation and trackers start on a common ground. The evaluation of an algorithm on the test dataset is done by submitting the results to a test server. The MOTChallenge website contains a leader board for each of the datasets, showing in separate pages models using the publicly provided detections and the ones using private detections. Online methods are also marked as so.

MOTA is the primary evaluation score for the MOTChallenge, but many other metrics are shown, including all the ones presented in section 2.2. As we will see, since the vast majority of MOT algorithms that use deep learning focus on pedestrians, the MOTChallenge datasets are the most widely used, as they are the most comprehensive ones currently available, providing more data to train deep models.

- **MOT15:** The first MOTChallenge dataset is 2D MOT 20152 [25] (often just called MOT15). It contains a series of 22 videos (11 for training and 11 for testing), collected from older datasets, with a variety of characteristics (fixed and moving cameras, different environments and lighting conditions, and so on) so that the models would need to generalize better in order to obtain good results on it. In total, it contains 11283 frames at various resolutions, with 1221 different identities and 101345 boxes. The provided detections were obtained using the ACF detector [26].

- **MOT16/17:** A new version of the dataset was presented in 2016, called MOT163 [9]. This time, the ground truth was made from scratch, so that it was consistent throughout the dataset. The videos are also more challenging, since they have a higher pedestrian density. A total of 14 videos are included in the set (7 for training and 7 for testing), with public detections obtained using the Deformable Part-based Model (DPM) v5 [34, 35], that they found to obtain better performance in detecting pedestrians on the dataset when compared to other models. This time the dataset includes 11235 frames with 1342 identities and 292733 boxes in total. The MOT17 dataset 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 [30], one from DPM and one from the Scale-Dependent Pooling detector (SDP) [29]. The trackers would then have to prove to be versatile and robust enough to get a good performance using different detection qualities.

- **MOT19:** Very recently, a new version of the dataset for the CVPR 2019 Tracking Challenge5 has been released, containing 8 videos (4 for training, 4 for testing) with extremely high pedestrian density, reaching up to 245 pedestrians per frame on average in the most crowded video. The dataset contains 13410 frames with 6869 tracks and a total of 2259143 boxes, much more than the previous datasets. While submissions for this dataset have only been allowed for a limited amount of time, this data will be the basis for the release of MOT19 in late 2019 [31].

- **KITTI:** While the MOTChallenge datasets focus on pedestrian tracking, the KITTI tracking benchmark6 [39, 40] allows for tracking of both people and vehicles. The dataset was collected by driving a car around a city and it was released in 2012. It consists of 21 training videos and 29 test ones, with a total of about 19000 frames (32 minutes). It includes detections obtained using the DPM7 and RegionLets8 [34] detectors, as well as stereo and laser information; however, as explained, in this survey we are only going to focus on models using 2D images. The CLEAR MOT metrics, MT, ML, ID switches and fragmentations are used to evaluate the methods. It is possible to submit results only for pedestrians or only for cars, and two different leader boards are maintained for the two classes

1.3.2 – Metrics

In order to provide a common experimental setup where algorithms can be equally tested and compared.

1.3.2.1 – Object detection metrics

Before going deeper with the most common metrics in the evaluation of object detection, the basic concepts need to be mentioned. When talking about object detection, the following definitions usually appear:

In the context of determining the validity of a detection (predicted mask or so-called bounding box), a supporting metric called *Intersection over Union* (also Jaccard Index) is needed.

1.3.2.1.1 - Intersection over Union (IoU)

In object detection problems, IoU evaluates the overlap between ground-truth mask/bounding gt and the predicted mask (pd). It is calculated as the area of intersection between gt and pd divided by the area of the union of the two. As follows:

$$\text{IoU} = \frac{\text{area}(gt \cap pd)}{\text{area}(gt \cup pd)}$$

- Pseudo code for IoU:

Algorithm 1: Computing Intersection Over Union (IoU)

INPUT $\boldsymbol{B_a}$& $\boldsymbol{B_b}$ (Box_a, Box_b COORDINATES) MATCHING THE FORMAT:

$(\boldsymbol{x(top-left)}, \boldsymbol{y(top-left)}, \boldsymbol{x(bottom-right)}, \boldsymbol{y(bottom-right)})$

OUTPUT THE INTERSECTION OVER UNION BETWEEN $\boldsymbol{B_a}$& $\boldsymbol{B_b}$.

PROCEDURE: $\boldsymbol{IoU} \leftarrow \boldsymbol{IoU}\ (\boldsymbol{b1}, \boldsymbol{b2})$

FIND THE AREA OF INTERSECTION

$\boldsymbol{xA}$ = MAX ($\boldsymbol{B_a}$ [0], $\boldsymbol{B_b}$ [0])

$\boldsymbol{yA}$ = MAX ($\boldsymbol{B_a}$ [1], $\boldsymbol{B_b}$ [1])

$\boldsymbol{xB}$ = MIN ($\boldsymbol{B_a}$ [2], $\boldsymbol{B_b}$ [2])

$\boldsymbol{yB}$ = MIN ($\boldsymbol{B_a}$ [3], $\boldsymbol{B_b}$ [3])

INTER-AREA ← MAX $(\boldsymbol{0}, \boldsymbol{xB} - \boldsymbol{xA} + \boldsymbol{1}) \times$ MAX $(\boldsymbol{0}, \boldsymbol{yB} - \boldsymbol{yA} + \boldsymbol{1})$

FIND THE AREA OF EACH BOUNDING BOX

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

$$AreaB = (B_b[2] - B_b[0] + 1) \times (B_b[3] - B_b[1] + 1)$$

$$IoU = Inter{-}Area / (AreaA + AreaB - Inter{-}Area)$$

END PROCEDURE

- Visual representation of the IoU (figure 1.8):

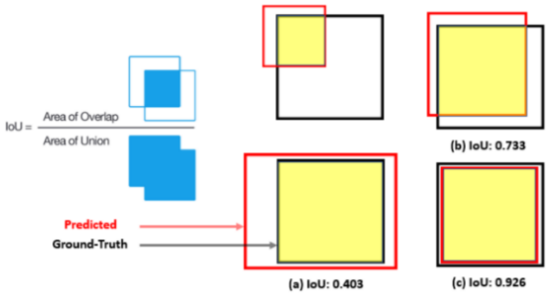


Figure 1.8 examples of computing IoU for Different bounding boxes.

1.3.2.1.2 – Common Terms in Object Detection

A. True positive (TP)

Is a correct detection. The condition is that the IoU must be above or equal to a given threshold. This threshold is usually defined in percentage to 50%, 75% or 95%. The results obtained by a system with these three thresholds can define its behaviour. For example, a given object detector can easily have good results at a 0,5 IoU but not so easily at a 0,95 IoU.

B. False positive (FP)

Is a false detection. The IoU of the detection must be below the threshold.

C. False negative (FN)

Absence of a detection of an object

D. True negative (TN)

It is not important but it is defined as all the possible bounding boxes that were correctly not detected. It is not used in metrics.

Metrics established by the Pascal VOC [20] challenge that uses *precision/recall* curve and *average precision* (Section 2.1.1.7).

1.3.2.1.3– Average Precision (AP)

- **Precision & Recall** (Figure 1.9)**:**

  - **Precision:** is the proportion of correct positive predictions.

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

- **Recall:** is the proportion of positive predictions with respect to all positives.

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

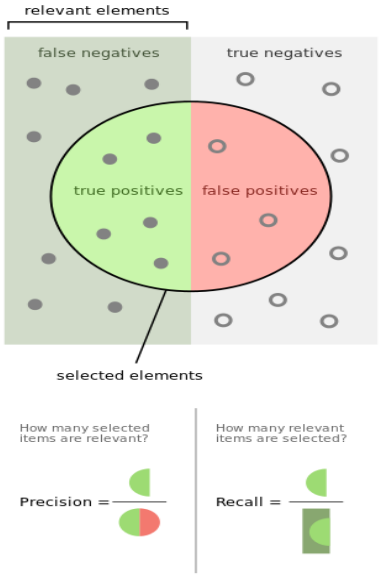


figure 1.9 Precision & Recall [97]

- **Precision/Recall Curve:** this curve plots the performance of an object detector as the confidence is changed for each object class (Figure 1.10)

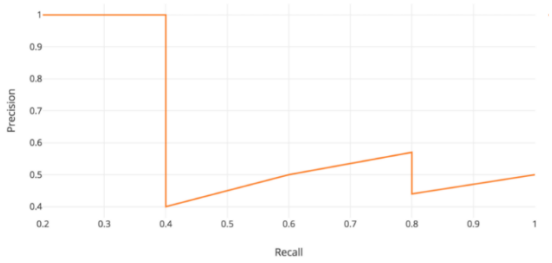


Figure 1.10 example of a Precision/Recall curve

- **Average Precision (AP):** The general definition for the Average Precision (AP) is finding the area under the precision-recall curve above.

$$\mathrm{AP} = \int_0^1 p(r)dr \qquad (1.3)$$

Precision and recall are always between 0 and 1. Therefore, AP falls within 0 and 1. Before calculating AP for the object detection, the "zigzags" pattern that does not permit an easy comparative between different curves (detectors). we often smooth out the "zigzag" pattern first. And this average can be in two main methods*:*

- *Eleven (11) - points interpolations: (method was used in Pascal VOC2008 competition [42])*

- *101 - Interpolating points (VOC2010–2012 [43])*

- **Eleven points interpolations:** Are defined as the mean precision at a set of eleven equally-spaced (figure 1.11) recall values ranging from 0 to 1 (Equation 2.1). The precision at each recall value is obtained by taking the maximum precision measured value for a method for which the corresponding recall is above r (Equation 2.2). This was the method used in Pascal VOC 2008.

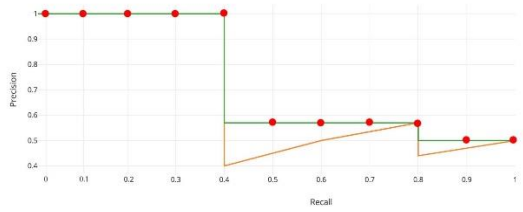


Figure 1.11 ex. of dividing the recall value from 0 to 1.0 into 11 points

$$AP = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} AP_r = \frac{1}{11} \sum_{r \in \{0.0,\ldots,1.0\}} p_{interp}(r) \quad (1.4)$$

Where

$$p_{interp}(r) = \max_{\tilde{r} \geq r} p(\tilde{r}) \quad (1.5)$$

- Simplified reading:

$$AP = \frac{1}{11} \times \left(AP_r(0) + AP_r(0.1) + \ldots + AP_r(1.0) \right) \quad (1.6)$$

- **101-points interpolations (Area Under Curve AUC):** in this case the mean precision is done interpolating through all recall points (Equation 2.3). The precision at each level r is obtained now taking the maximum precision which has a recall value greater or equal than the recall value at the level r + 1 (Equation 2.4). This method of interpolation is used in Pascal VOC metrics from the year 2010 onwards.

$$\mathrm{AP} = \Sigma \left(r_{n+1} - r_n \right) p_{interp}(r_{n+1}) \quad (1.7)$$

$$p_{interp}(r_{n+1}) = \max_{\tilde{r} \geq r_{n+1}} p(\tilde{r}) \quad (1.8)$$

With this interpolation the Area Under Curve (AUC) is obtained exact

- Graphical representation of the calculations (figure 1.12)

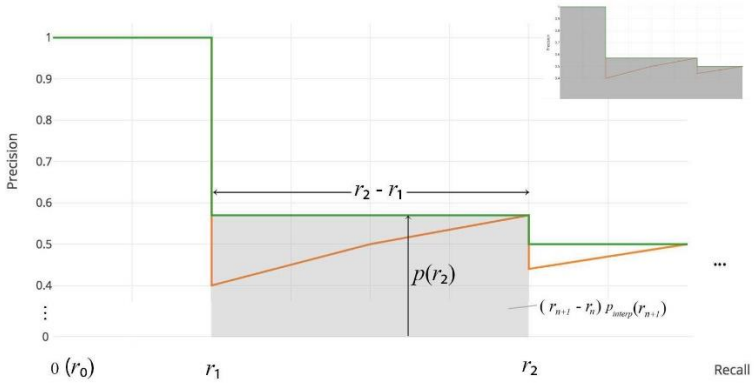


Figure 1.12 All points interpolation [96]

- Pseudo code for AP:

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

INPUT LIST OF (GROUND-TRUTH) $\boldsymbol{GTi}$ AND PREDICTED $\boldsymbol{Di}$ (BOUNDING BOX OF A DETECTION) FOR EVERY FRAME
$\boldsymbol{D}$ DETECTION LIST
IN THE SEQUENCE ($\boldsymbol{i}$ REPRESENT THE FRAME NUMBER).
OUTPUT THE AVERAGE PRECISION AP (AP)
$\boldsymbol{confscore}$ REPRESENTS CONFIDENCE SCORE OF A DETECTION
$\boldsymbol{confthreshold}$ REPRESENTS CONFIDENCE THRESHOLD
$\boldsymbol{TP}$ REPRESENTS TRUE POSITIVE LIST
$\boldsymbol{FP}$ REPRESENTS FALSE POSITIVE LIST

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

FOR $\boldsymbol{f}$ IN LENGTH ($\boldsymbol{D}$):

FOR $\boldsymbol{d}$ IN $\boldsymbol{Df}$:

$\boldsymbol{d(confscore)} = \boldsymbol{max(IoU(GTf,d))}$

IF $\boldsymbol{d(confscore)} \geq \boldsymbol{confthreshold}$:

$\boldsymbol{TP(list)} \leftarrow \boldsymbol{1}$
$\boldsymbol{FP(list)} \leftarrow 0$

$\boldsymbol{else}$:

$$\boldsymbol{TP(list)} \leftarrow \mathbf{0}$$
$$\boldsymbol{FP(list)} \leftarrow \mathbf{1}$$

$\boldsymbol{TPs}$ & $\boldsymbol{FPs}$ ← SORT $\boldsymbol{TP(list)}$ & $\boldsymbol{FP(list)}$ IN DESCENDING ORDER ACCORDING TO THE CONFIDENCE SCORES.

$$\boldsymbol{Precision} \leftarrow \frac{\sum \text{TPs}}{\sum \text{TPs} + \sum \text{FPs}}$$

$$\boldsymbol{Recall} \leftarrow \frac{\sum \text{TPs}}{\sum_F \text{GT}_F}$$

AP ← FORM $\boldsymbol{Recall} - \boldsymbol{Precision}$ PLOT AND COMPUTE THE ESTIMATED $\mathbf{area1}$

END PROCEDURE

- **Mean Average Precision (mAP) COCO dataset:** 101-point interpolation AP is applied in calculation, AP is the average over multiple IoU, (e.g., AP@[.5:.95] corresponds to the average AP for IoU from 0.5 to 0.95 with a step size of 0.05). For the COCO competition [21], AP is the average over 10 IoU levels on 80 categories.

1.3.2.2 – Object Tracking metrics

A group of metrics have been established as standard, and they are used in almost every work. The most relevant ones are metrics defined by Wu and Nevatia [12], under the name of CLEAR MOT metrics [13], and recently in 2016 the ID metrics [14]. These sets of metrics aim to reflect the overall performance of the tested models; Therefore, those metrics are defined as follows:

1.3.2.2.1 - Classical metrics

These metrics, defined by Wu and Nevatia [12], highlight the different types of errors a MOT algorithm can make. In order to show those problems, the following values are computed (figure 1.13):

- Mostly Tracked (MT) trajectories: number of ground-truth trajectories that are correctly tracked in at least 80% of the frames.
- Fragments: trajectory hypotheses which cover at most 80% of a ground truth trajectory. Observe that a true trajectory can be covered by more than one fragment.
- Mostly Lost (ML) trajectories: number of ground-truth trajectories that are correctly tracked in less than 20% of the frames.
- False trajectories: predicted trajectories which do not correspond to a real object (i.e., to a ground truth trajectory).
- ID switches: number of times when the object is correctly tracked, but the associated ID for the object is mistakenly changed.

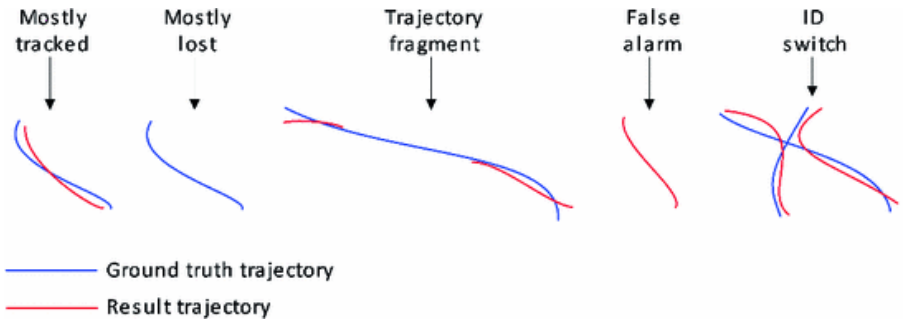


Figure 1.13 Classical tracking evaluation criteria [12]. The blue trajectories indicate ground-truth tracklet, while the red ones are predicted.

1.3.2.2.2– CLEAR MOT metrics

These metrics appeared for *Classification of Events, Activities and Relationships* the workshops held in 2006[15] and 2007[16] and they were jointly organized by the European CHIL project, the U.S. VACE project, and the National Institute of Standards and Technology (NIST). Those metrics are MOTA (Multiple Object Tracking Accuracy) and MOTP (Multiple Object Tracking Precision). In our case, we are going to be focusing on 2D tracking with single camera, the most used metric to decide whether an object and a prediction are related or not is *Intersection over Union* (IoU) of bounding boxes, as it was the measure established in the presentation paper of MOT15 dataset [17]. Specifically, the mapping between ground truth and hypotheses is established as follows: if the ground truth object oi and the hypothesis hj are matched in frame $t-1$, and in frame t the $IoU(oi, hj) \geq 0.5$, then oi and hj are matched in that frame, even if there exists another hypothesis hk such that $IoU(oi, hj) < IoU(oi, hk)$, considering the continuity constraint. After the matching from previous frames has been performed, the remaining objects are tried to be matched with the remaining hypotheses.

The MOTA score is then defined as follows:

$$MOTA = 1 - \frac{(FN + FP + IDSW)}{GT} \quad \in (-\infty, 1]$$

let's define the score terms:

- ***False Negative (FN):*** The ground truth bounding boxes oi that cannot be associated with a hypothesis hj are counted as false negatives

- ***False Positive (FP):*** and the hypotheses hj that cannot be associated with a real bounding box are marked as false positives

- ***IDSW:*** every time a tracked ground truth object ID is incorrectly changed during the tracking duration is counted as an ID switch

- ***GT:*** is the number of ground truth boxes (TP+FN)

Note: the score can be negative, as the algorithm can commit a number of errors greater than the number of ground truth boxes. Usually, instead of reporting MOTA, it is common to report the percentage MOTA, same expression but expressed as percentage.

The MOTP is expressed as follows:

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t}$$

where ct denotes the number of matches in frame t, and dt,i is the bounding box overlap between the hypothesis i with its assigned ground truth object. It is important to note that this metric takes few information about tracking into account, and rather focuses on the quality of the detections.

1.3.2.2.3– Identification (ID) metrics

The main problem of MOTA score is that it takes into account the number of times a tracker makes an incorrect decision. such as an ID switch, in some cases we would be more interested in a tracker who can track an object for a longer period of time for this reason, E. Ristani et al. [18] in 2016 proposed new metrics that are supposed to complement CLEAR MOT metrics. In contrast to the MOTA, the ID metrics perform matching *globally* not like the MOTA which matches Ground-truth with the perdition *frame-by-frame*. 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 1.14) consists of two sets, Vt which include a node for each GT tracklet $\{v1, v2, \ldots\}$ and $\mathcal{F}ci$ + false positive node for each computed tracklet. The second set is Vc, include a node for each predicted tracklet $\{c1, c2, \ldots\}$ and $\mathcal{F}vi$ − false negative for each GT one. With this setup, four possible pairs are present. The edge cost is set as a binary, that is for counting purpose (More information can be found in [18]).

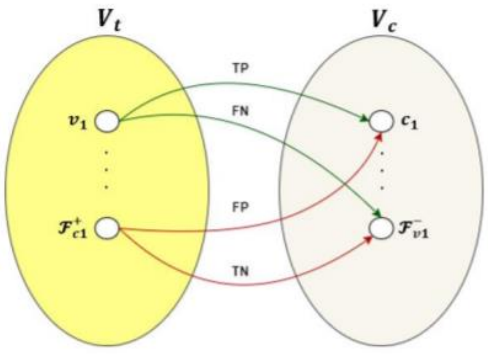


Figure 1.14 Example ID Bipartite graph with one tracklet for demonstration [18]

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 $\mathcal{F}vi-$ edges.

- IDFP: Sum of the weights from the selected $\mathcal{F}ci+$ edges.

Another three important measures using the previous scores as follows:

- Identification Precision (IDP) $$IDP = \frac{IDTP}{IDTP + IDFP}$$
- Identification Recall (IDR) $$IDR = \frac{IDTP}{IDTP + IDFN}$$
- Identification F1 IDF1) $$IDF1 = \frac{2}{\frac{1}{IDP} + \frac{1}{IDR}}$$

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

Chapter 2 : Tracking-by-Detection using Deep Learning

Introduction

While many works have used as input to their algorithm's dataset-provided detections generated by various detectors (for example Aggregated Channel Features [17] for MOT15 [25] or Deformable Parts Model [18] for MOT16 [9]), there have also been algorithms that integrated a custom detection step, that often contributed to improve the overall tracking performance by enhancing the detection quality. Before we jump to the tracking phase, and because the detection step is critical and it is a very important in the tracking process, we first jump to the Detection phase and see the most of the effective object detecting algorithms that's solves this problem.

2.1- Object detectors

A modern detector is usually composed of two parts, a backbone which is pre-trained on ImageNet [10] and a head which is used to predict classes and bounding boxes of objects. Detectors who run on a GPU platform the back bone could be like VGG16[84] orDarknet53[44], and ones who run on CPU platform like MobileNet [46], shuffleNet [47], etc. Object detectors developed in recent years often insert some layers between backbone and head, and these layers are usually used to collect feature maps from different stages. We can call it the neck of an object detector. Usually, a neck is composed of several bottom-up paths and several top-down paths like Feature Pyramid Network (FPN) [85], or Path Aggregation Network (PAN) [86].

to sum up, an ordinary object detector is composed of several parts

- **Input**: Image, Patches, Image Pyramid
- **Backbones**: VGG16 [68], ResNet-50 [26], SpineNet [12], EfficientNet-B0/B7 [75], CSPResNeXt50 [81], CSPDarknet53 [81]
- **Neck**:
  - **Additional blocks**: SPP [25], ASPP [5], RFB [47], SAM [85]
  - **Path-aggregation blocks**: FPN [44], PAN [49], NAS-FPN [17], Fully-connected FPN, BiFPN [77], ASFF [48], SFAM [98]
- **Heads:**:
  - **Dense Prediction (one-stage)**:
    - RPN [64], SSD [50], YOLO [61], RetinaNet [45] (anchor based)
    - CornerNet [37], CenterNet [13], MatrixNet [60], FCOS [78] (anchor free)
  - **Sparse Prediction (two-stage)**:
    - Faster R-CNN [64], R-FCN [9], Mask R-CNN [23] (anchor based)
    - RepPoints [87] (anchor free)

Figure 2.1. summary of object detector composition from [75]

2.1.1- Backbone networks

Backbone network is acting as the basic feature extractor for object detection task which takes images as input and outputs feature maps of the corresponding input image. Most of backbone networks for detection are the network for classification task taking out the last fully connected layers. The improved version of basic classification network is also available. For instance, Lin et al. [42] add or subtract layers or replace some layers with special designed layers. To better meet specific requirements, some works [41] [43] utilize the newly designed backbone for feature extraction.

Towards different requirements about accuracy vs. efficiency, people can choose deeper and densely connected backbones, like DarkNet-53 [44] (Figure 2.2) used in YOLO as a backbone (You Only Look Once), ResNet [45] that used in SSD (Single Shot Detector) or light weight backbones like MobileNet [46] (Figure 2.3), ShuffleNet [47]. When applied to mobile devices, lightweight backbones can meet the requirements. In order to meet the needs of high precision and more accurate applications, complex backbones are a must. On the other hand, real-time acquirements like video or webcam require not only high processing speed but high accuracy [49], which need well-designed backbone to adapt to the detection architecture and make a trade-off between speed and accuracy. The newly high performance classification networks can improve precision and reduce the complexity of object detection task. This is an effective way to further improve network.

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

Figure 2.2 Darknet-53 Architecture [44]

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
$5\times$ Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2.3 MobileNetV1 Architecture [46]

With the advancement of deep learning and the continuous improvement of computing power, great progress has been made in the field of general object detection. In 2014 the first CNN-based object detector R-CNN [48] was proposed, and then a series of significant contributions have been made which promote the development of general object detection by a large margin. We introduce some state-of-the-art object detection algorithms.

2.1.2 – Two stage Detectors

Identify regions where might be object then specify it

2.1.2.1 – R-CNN

Region based CNN detector As Girshick et al. [50] propose R-CNN which can be used in object detection tasks, their works are the first to show that a CNN could lead to dramatically higher object detection performance on PASCAL VOC datasets [20] than those systems based on simpler HOG-like features. Deep learning method is verified effective and efficient in the field of object detection.

R-CNN detector consists of four modules:

1st - generates category-independent region proposals, the authors adapt a selective search method then a CNN is used to extract a 4096-dimensional feature vector from each region proposal. Because the fully connected layer needs input vectors of fixed length (The authors adopt a fixed 227×227 pixel as the input size of CNN)

2nd - extracts a fixed-length feature vector from each region proposal. consists of five convolutional layers and two fully connected layers. And all CNN parameters are shared across all categories.

3rd - is a set of class-specific linear SVMs to classify the objects in one image, each category trains category-independent SVM which does not share parameters between different SVMs.

4th – is a bounding-box regressor for precisely bounding-box prediction [51].

- Inputs: sub-regions of the image corresponding to objects.
- Outputs: New bounding box coordinates for the object in the sub-region.

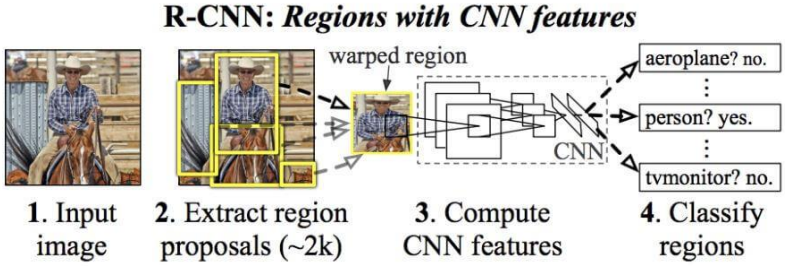


Figure 2.4 RCNN architecture

❖ **Limitations:** Training an RCNN model is expensive and slow duo to

- Extracting 2,000 regions for each image based on selective search
- Extracting features using CNN for every image region. Suppose we have N images, then the number of CNN features will be N*2,000
- The entire process of object detection using RCNN has three models:

  - CNN for feature extraction
  - Linear SVM classifier for identifying objects
  - Regression model for tightening the bounding boxes

2.1.2.2 – Fast R-CNN

A year later in 2015, Ross Girshick proposed a faster version of R-CNN, called Fast R-CNN [52] (figure 2.5). Because R-CNN performs a ConvNet forward pass for each region proposal without sharing computation, R-CNN takes a long time on SVMs classification. Fast RCNN extracts features from an entire input image and then passes the region of interest (RoI) pooling layer to get the fixed size features as the input of the following classification and bounding box regression fully connected layers. The features are extracted from the entire image once and are sent to CNN for classification and localization at a time. Compared to R-CNN which inputs each region proposals to CNN, a large amount of time can be saved for CNN processing and large disk storage to store a great deal of features can be saved either in Fast R-CNN.

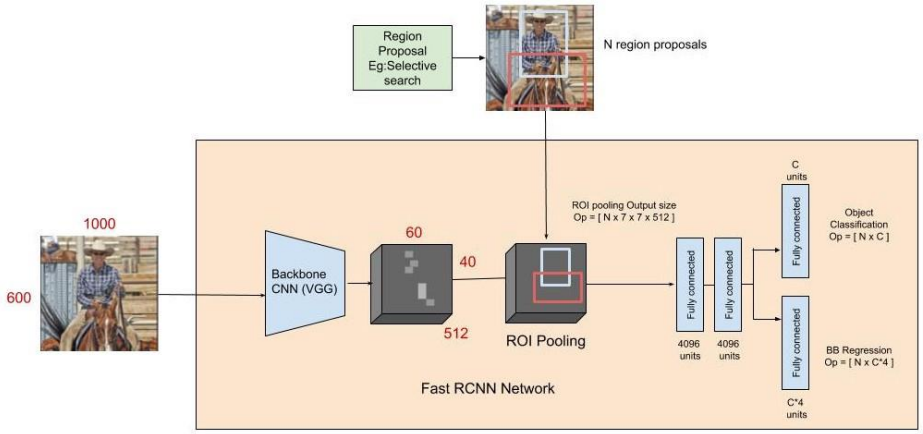


Figure 2.5 Fast R-CNN architecture

- **Limitations:** Both of the above algorithms (R-CNN & Fast R-CNN) uses selective search [53] to find out the region proposals. Selective search is a slow and time-consuming process affecting the performance of the network. Therefor is not the ideal algorithm for online applications

2.1.2.3 – Faster R-CNN

Three months after Fast R-CNN was proposed, Faster R-CNN [54] (figure 2.6) further improves the region-based CNN baseline. Fast R-CNN uses selective search to propose RoI, as we mentioned earlier is slow and needs the same running time as the detection network. Faster R-CNN replaces it with a novel RPN (region proposal network) that is a fully convolutional network to efficiently predict region proposals with a wide range of scales and aspect ratios. RPN accelerates the generating speed of region proposals because it shares fully-image convolutional features and a common set of convolutional layers with the detection network. a novel method for different sized object detection is that multi-scale anchors are used as reference. The anchors can greatly simplify the process of generating various sized region proposals with no need of multiple scales of input images or features. On the outputs (feature maps) of the last shared convolutional layer, sliding a fixed size window (3 × 3), the centre point of each feature window is relative to a point of the original input image which is the center point of k (3 × 3) anchor boxes. The authors define anchor boxes have 3 different scales and 3 aspect ratios. The region proposal is parameterized relative to a reference anchor box. Then they measure the distance between predicted box and its corresponding ground truth box to optimize the location of the predicted box. Experiments [54] indicated that Faster R-CNN has greatly improved both precision and detection efficiency

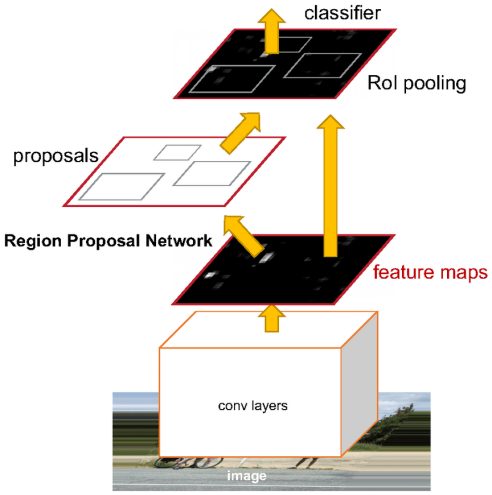


Figure 2.6 Faster R-CNN architecture same as Fast R-CNN but instead of selective search for region proposal they used Region Proposal Network

2.1.3– One stage Detectors

Identify regions and classify it at once.

2.1.3.1 – YOLO

YOLO [49] stands for you only look once, proposed by Redmon et al. after Faster RCNN [54]. The main contribution is real-time detection of full images and webcam. Firstly, the downside about this pipeline, it only predicts less than 100 bounding boxes per image while Fast R-CNN using selective search predicts 2000 region proposals per image. Secondly, YOLO frames detection as a regression problem, so a unified architecture can extract features from input images directly.

This model, from the "single-shot networks family" (one stage detector), uses a different approach. This network divides the image into regions and predicts the bounding box and probabilities of each region. These are then weighted with the probabilities to obtain the definitive detections (Figure 2.7). This performs, as the authors indicate, a hundred times faster than Fast R-CNN [54] maintaining a similar accuracy.

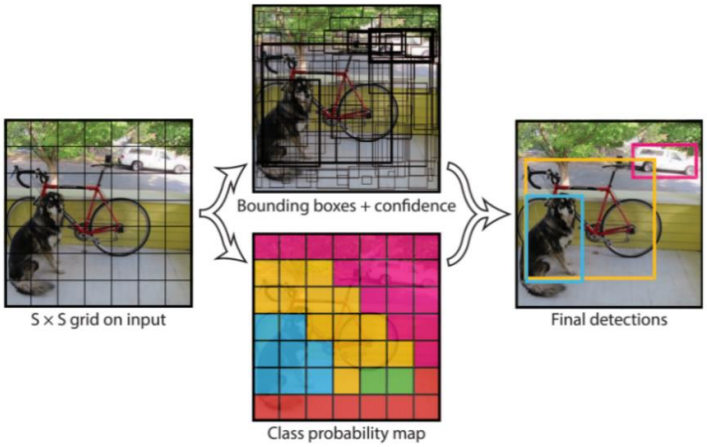


Figure 2.7. the system models detection as a regression problem.It divides the image into an S × S grid and for each grid cell predicts B bounding boxes, confidence for those boxes, and C class probabilities. These predictions are encoded as an S × S × (B ∗ 5 + C) tensor. From [49]

2.1.3.2 – YOLOv4

YOLOv4: Our study is going to be focusing on the revolutionary update of YOLO, the "YOLOv4" [75] has an incredibly high performance for a very high FPS which is very suitable for Realtime application (like in our case) this was a major improvement from previous object detection models which only had either high performance or high inference speeds.

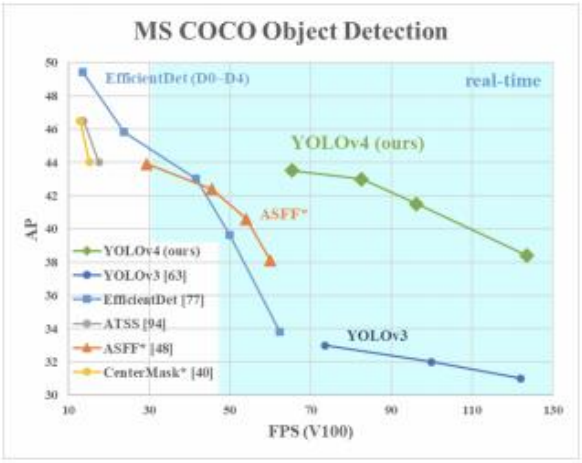


Figure 2.8. Comparison of the proposed YOLOv4 and other state-of-the-art object detectors. YOLOv4 runs twice faster than EfficientDet with comparable performance. Improves YOLOv3's AP and FPS by 10% and 12%, respectively [75].

YOLOv3 [76] was introduced in 2018 as an "Incremental Improvement" Stating that it was simply a bit better than YOLOv2, but not much changed.

	backbone	AP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
Two-stage methods							
Faster R-CNN+++ [5]	ResNet-101-C4	34.9	55.7	37.4	15.6	38.7	50.9
Faster R-CNN w FPN [8]	ResNet-101-FPN	36.2	59.1	39.0	18.2	39.0	48.2
Faster R-CNN by G-RMI [6]	Inception-ResNet-v2 [21]	34.7	55.5	36.7	13.5	38.1	52.0
Faster R-CNN w TDM [20]	Inception-ResNet-v2-TDM	36.8	57.7	39.2	16.2	39.8	**52.1**
One-stage methods							
YOLOv2 [15]	DarkNet-19 [15]	21.6	44.0	19.2	5.0	22.4	35.5
SSD513 [11, 3]	ResNet-101-SSD	31.2	50.4	33.3	10.2	34.5	49.8
DSSD513 [3]	ResNet-101-DSSD	33.2	53.3	35.2	13.0	35.4	51.1
RetinaNet [9]	ResNet-101-FPN	39.1	59.1	42.3	21.8	42.7	50.2
RetinaNet [9]	ResNeXt-101-FPN	**40.8**	**61.1**	**44.1**	**24.1**	**44.2**	51.2
YOLOv3 608 × 608	Darknet-53	33.0	57.9	34.4	18.3	35.4	41.9

Figure 2.9. table shows that YOLOv3 is much better than SSD variants and comparable to state-of-the-art models on the AP50 metric [76].

Architecture

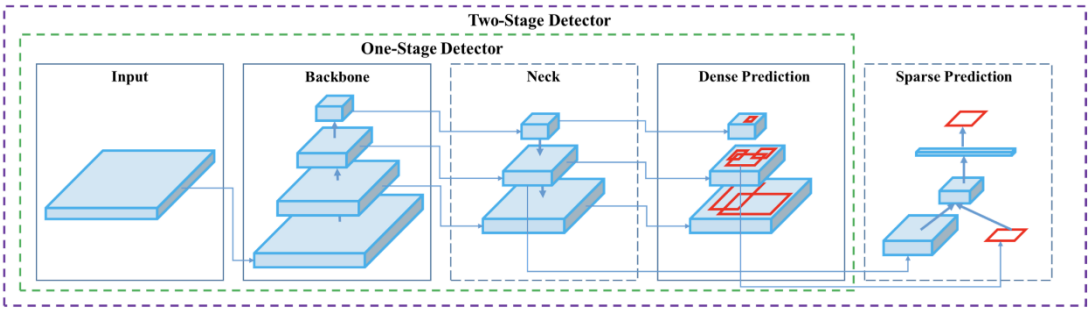


Figure 2.10. Object detector in general [75]

a) Backbone:

DarkNet53 [44] (used in YOLOv3)**,** and CSPDarkNet53 [87] (used in YOLOv4)

CSP stands for *Cross-Stage-Partial* connections. The idea here is to separate the current layer into 2 parts, one that will go through a block of convolutions, and one that won't. Then, we aggregate the results. Here's an example with DenseNet:

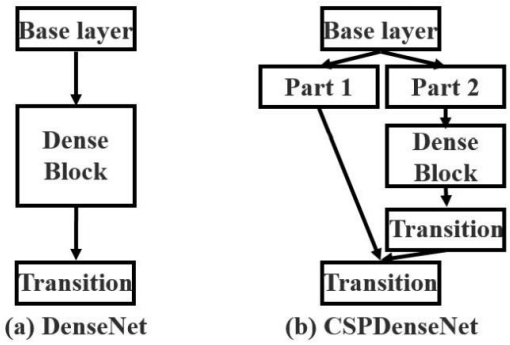


Figure 2.11. DenseNet vs CSPDenseNet comparison [85]

Explaining the architecture work flow will be beyond this master thesis scope, so let's just know that the backbones used for features extraction purposes, see for more details [85]

b) Neck:

The purpose of the neck block is to add extra layers between the backbone and the head (**dense prediction block**). You might see that different feature maps from the different layers used.

YOLOv4 authors used a modified version of the PANet [86] (Path Aggregation Network)

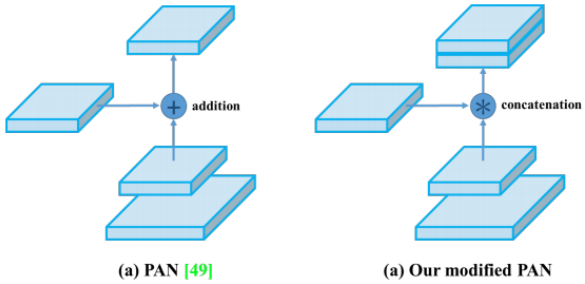


Figure 2.12 Comparison of PAN and modified PANet by YOLOv4 authors [75]

Another technique used is Spatial Attention Module (SAM) [86]. Attention mechanisms have been widely used in deep learning, and especially in recurrent neural networks. It refers to focusing on a specific part of the input.

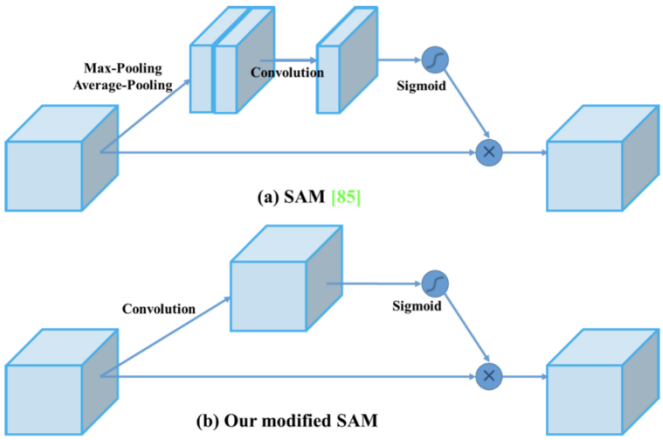


Figure 2.13 Comparison of SAM and modified SAM by YOLOv4 authors [75]

And finally, Spatial Pyramid Pooling (SPP) [88], used in R-CNN [50] networks and numerous other algorithms, is also used here.

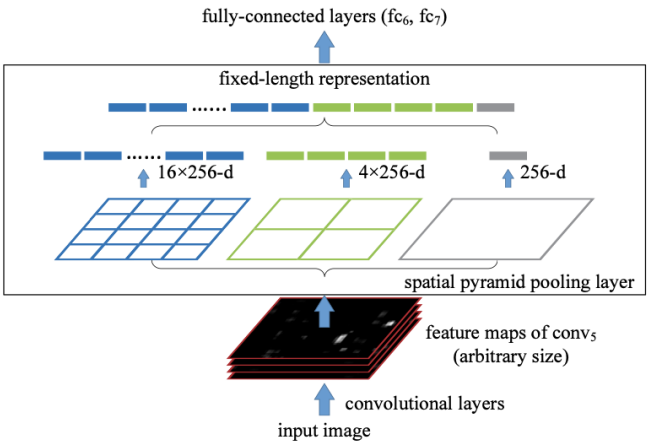


Figure 2.14 Spatial Pyramid Pooling Network [88]

Some techniques exist for adding information in a layer, a bit like a ResNet would do. YOLOv4 uses a modified Path Aggregation Network, a modified Spatial Attention Module, and Spatial Pyramid Pooling.

c) Head:

The head block is used to (I) Locate Bounding Box, (II) classify what's inside each box.

Here, we have the same process as in YOLOv3 [76]. The network detects the bounding box coordinates (x, y, w, h) as well as the confidence score for a class. This technique is anchor-based.

1. Bounding box prediction:

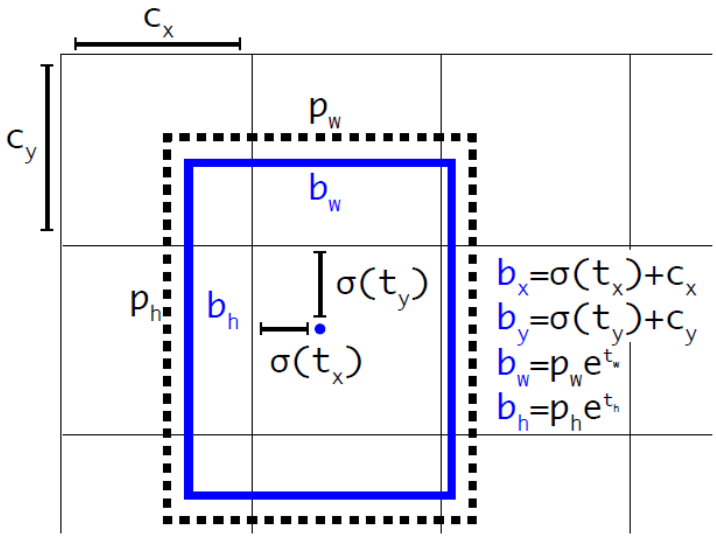


Figure 2.15. Bounding Box Prediction, Predicted Box (Blue), Prior Box (Black Dotted) [76]

The network predicts 4 coordinates for each bounding box, tx, ty, tw, th. If the cell is offset from the top left corner of the image by (cx, cy) and the bounding box prior has width and height pw, ph, then the predictions correspond to:

$$
\begin{aligned}
b_x &= \sigma(t_x) + c_x \\
b_y &= \sigma(t_y) + c_y \\
b_w &= p_w e^{t_w} \\
b_h &= p_h e^{t_h}
\end{aligned}
\qquad (2.1)\ [76]
$$

During training they use sum of squared error loss [77]. If the ground truth for some coordinate prediction is tˆ *, the gradient is the ground truth value (computed from the ground truth box) minus the output prediction: tˆ * − t*. This ground truth value can be easily computed by inverting the equations above (2.1)

It also predicts an objectness score for each bounding box using logistic regression [77]. This should be 1 if the bounding box prior overlaps a ground truth object by more than any other bounding box prior. If the bounding box prior is not the best but does overlap a ground truth object by more than

some threshold the production is ignored, following the faster R-CNN [54]. they used the threshold of 0.5. Unlike [54] YOLOv3 system only assigns one bounding box prior for each ground truth object. If a bounding box prior is not assigned to a ground truth object it incurs no loss for coordinate or class predictions, only objectness

2. Class Prediction:

Each box predicts the classes, the bounding box may contain using multilabel classification. they used independent logistic classifiers [78]. During training they used binary cross-entropy loss for the class predictions [79]. This formulation helps when the goal is to move to more complex domains like the Open Images Dataset [23]. Because in this dataset (which been used in this master thesis project), there are many overlapping labels (i.e., Woman and Person). Using a SoftMax imposes the assumption that each box has exactly one class which is often not the case. A multilabel approach better models the data.

3. Prediction across scales:

Unlike YOLOv1, the YOLOv4 (and YOLOv3) has 3 different scales are used for prediction, the feature extraction from these scales using similar concept of feature pyramid network [80], several convolutional layers are added to the base feature extractor (from darknet-19 [81] to become darknet53 [44] layers for YOLOv3 than CSPDarknet [46] for YOLOv4), the last of these layers predicts the bounding box, objectness and class predictions

On COCO dataset [21], **3 boxes at each scale.** Therefore, the output tensor is $S \times S \times$ **[3× (4+1+80)],** i.e., **4 bounding box offsets, 1 objectness prediction, and 80 class predictions (car, cat, dog, …etc),** where (S x S) stands for grid of the image (see Figure 2.7). the output vector will be holding the bounding box coordinates and probability classes, and a post-processing technique such as the *non-maxima suppression* (NMS) [82]

The authors of YOLOv4 have worked on techniques to improve the accuracy of the model while training and in post-processing. These techniques are called *bag-of-freebies and bag-of-specials*. We can't cover everything. Duo to the scope of this master thesis

4. Bag of freebies (BoF):

is a set of techniques that help during training without adding much inference time. Some popular techniques include data augmentation, random cropping, shadowing, dropout, etc.

- **BoF for the backbone:** CutMix and Mosaic data augmentation, Drop Block regularization, Class label smoothing.

- **BoF for the detector:** CIoU-loss [75], CmBN[75], DropBlock regularization [83], Mosaic data augmentation, self-adversarial training (SAT) [75], eliminate grid sensitivity, using multiple anchors for a single ground-truth sample, cosine annealing scheduler, optimizing hyperparameters, random training shapes.

5. **Bag of specials (BoS):** another family of techniques. Unlike BoF, they change the architecture of the network and thus might augment the inference cost a bit. We already saw SAM, PAN, and SPP, which all belong to this family.

    - **BoS for the backbone:** Mish activation, cross-stage partial connections (CSP), multi-input weighted residual connections (MiWRC)
    - **BoS for detector:** Mish activation, SPP-block, SAM-block, PAN path-aggregation block, DIoU-NMS

In short these are advanced data augmentation techniques made by the YOLOv4 authors, to make YOLOv4 a *state-of-the-art* object detector. And data augmentation techniques will be explained in details in the implementation chapter

2.2 - MOT Trackers

Before we speak about the algorithms that are available for MOT problems, let 'see an overview about MOT tracking, is it very difficult to classify one particular MOT method into a distinct category

2.2.1-MOT Categorization

According to Wenham Luo et al [89] we can conduct this in three criteria:

a) Initializing method
b) Processing mode
c) Type of output

a) Initializing method: Most existing MOT works can be grouped into two sets [42], depending on how objects are initialized

1. **Detection-Based Tracking (DBT):** As shown in Figure 2.16 (top), objects are first detected and then linked into trajectories. This strategy is also commonly referred to as "tracking-by-detection". Given a sequence, type-specific object detection or motion detection (based on background modelling) [43], [44] is applied in each frame to obtain object hypotheses, then (sequential or batch) tracking is conducted to link detection hypotheses into trajectories. There are two issues worth noting. First, since the object detector is trained in advance, the majority of DBT focuses on specific kinds of targets, like in our case is going to be more focused on pedestrian, and the performance of DBT highly depends on the performance of the employed object detector
2. **Detection-Free tracking (DFT):** As shown in Figure 2.16 (bottom), DFT [45], [46], [47], [48] requires manual initialization of a fixed number of objects in the first frame, then localizes these objects in subsequent frames. DBT is more popular because new objects are discovered and disappearing objects are terminated automatically. DFT cannot deal with the case that objects appear. However, it is free of pre-trained object detectors.

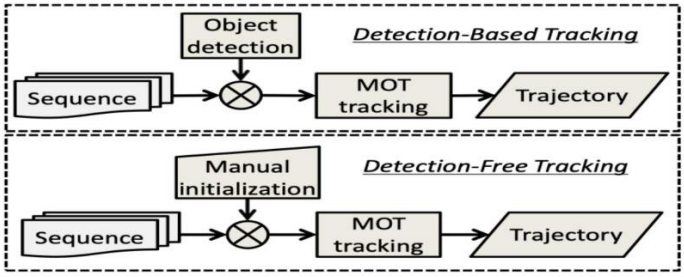


Figure 2.16. flow of Detection-Based Tracking (DBT) vs Detection-Free Tracking (DFT)

Adapting from B. Yang and R. Nevatia [55] we see the major differences between DBT and DFT as follows:

Item	**DBT**	**DFT**
Initialization	automatic, imperfect	manual, perfect
Number of objects	Varying	Fixed
Applications	specific type of objects (In most cases like pedestrian in our case)	any type of objects
Advantages	ability to handle varying number of objects	free of object detector
Drawbacks	performance depends on object detection	manual initialization

Table 2.1. comparison between DBT and DFT

b) Processing mode: MOT can also be divided into two, *online* Tracking and *offline* tracking, the difference lays in whether or not use future observations from the future frames when handling the current frame. Online tracking method (which is going to be using in this master thesis) which only relay on the past information available up to the current frame while offline (batch) tracking approaches use observation from both in the past and in the future (see figure 2.17)

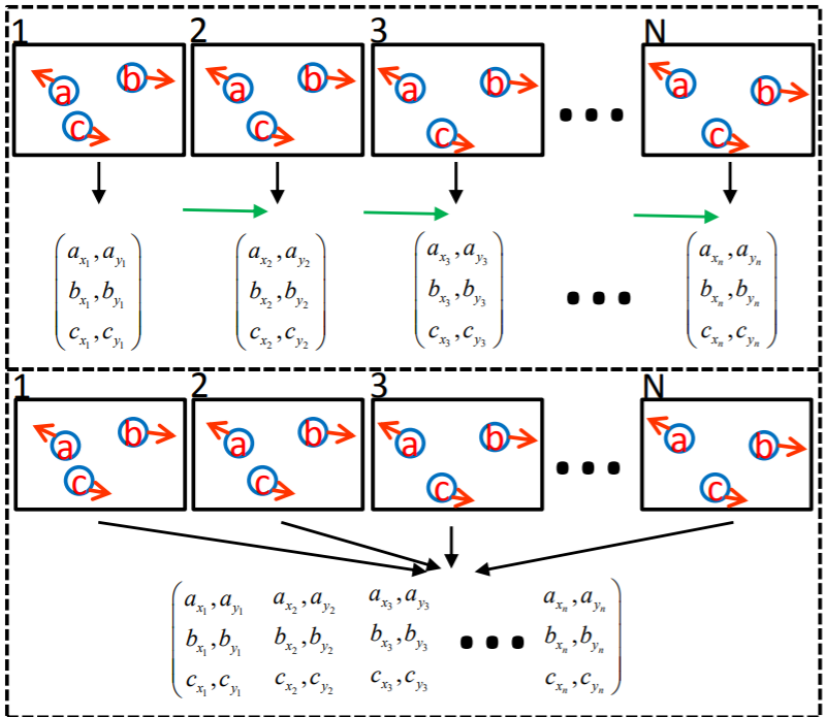


Figure 2.17 online tracking (top) vs offline or batch tracking (bottom)

b.1) Online Tracking: In online tracking [56], [57]. The image sequence is handled in a step-wise manner, thus online tracking is also named as sequential tracking. An illustration is shown in Figure 2.17 (top), with three objects (different circles) a, b, and c. The green arrows represent observations in the past. The results are represented by the object's location and its ID. Based on the up-to-time observations, trajectories are produced on the fly

b.2) Offline Tracking: Offline tracking [53], [54], uses a batch of frames to process the data. As shown in Figure 2.17 (bottom), observations from all the frames are required to be obtained in advance and are analysed jointly to estimate the final output. Note that, due to computational and memory limitation, usually we split the data into shorter video clips, and infer the results hierarchically or sequentially for each batch (series of frames). According to Wenhan [61] we can list the differences between the two-processing mode (table 2.2).

Item	Online tracking	Offline tracking
input	Up-to-time observations	All observations
Methodology	Gradually extend existing trajectories with current observations	Link observations into trajectories
Advantages	Suitable for online tasks like live streaming	Obtain global optimal solution theoretically
Drawbacks	Suffer from shortage of observation	Delay in outputting final results

Table 2.2 Comparison between online and offline tracking

c) Type of output: This criterion classifies MOT methods into deterministic ones and probabilistic ones, depending on the randomness of output. The output of deterministic tracking is constant when running the methods multiple times. While output results are different in different running trials of probabilistic tracking methods

2.2.2 - Online tracking algorithms

And because the goal of this master Thesis is building a Robust and Realtime tracking which means that we only going to relay just on the current and previous frames, so we have to choose carefully the algorithms that fit to this purpose. There's a lot of online tracking algorithms, but in this master thesis we are going to be focusing on the most effective and state-of-the-art and open-source tracker Algorithm called *DeepSORT* [59], but firstly we are going to speak about *SORT* [58] (simple online Realtime tracking)

2.2.2.1 – Simple Online Realtime tracker (SORT)

SORT is a lightweight, open-source multiple object tracker that his main components are the Kalman filter [62] and Hungarian algorithm [63]. Object locations are frame-by-frame using a linear constant velocity model. Each tracklet maintains a state vector containing the target's estimated position and velocity. During the affinity stage (testing if two identities are the same), a cost matrix is constructed by computing the IoU distance (*explained in chapter 1, Object Detection metrics*) between each detection and tracklet. Certain assignments are marked impossible if the IoU distance for the assignment is greater than a certain threshold. Optimal assignments are generated from the cost matrix using the Hungarian algorithm. For each matched (detection, track) pair, the detected bounding box is used to update the position components of the target state, and the velocity components are estimated with a Kalman filter. This simple framework does not involve neural networks beyond the detection stage, making SORT an extremely fast multiple object tracker. However, this simplicity also hinders the accuracy of SORT, especially with respect to the general tracking issues such as occlusions.

2.2.2.1.1– Detection

Detection quality is identified as a key factor influencing tracking performance, where changing the detector can improve tracking by up to 18.9% [58]. Despite only using a rudimentary combination of familiar techniques such as the Kalman Filter and Hungarian algorithm for the tracking components, this approach achieves an accuracy comparable to state-of-the-art online trackers. Furthermore, due to the simplicity of our tracking method, the tracker updates at a rate of 260 Hz which is over 20x faster than other state-of-the-art trackers. (Figure 2.18)

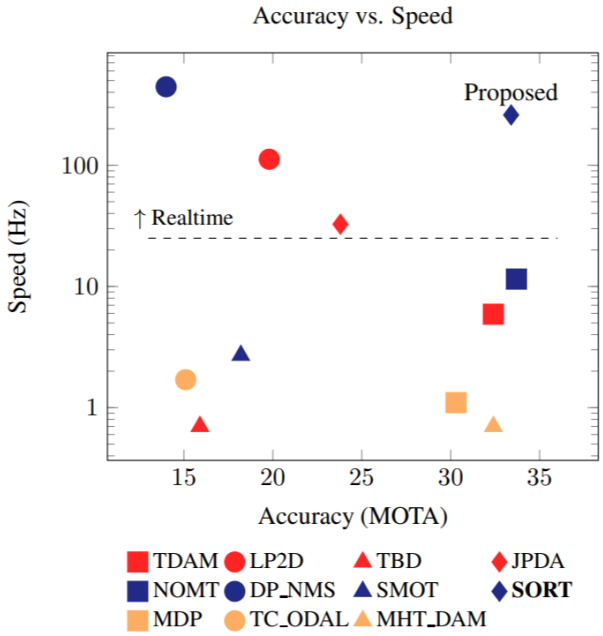


Figure 2.18 Benchmark performance of the (SORT) method in relation to several baseline trackers [64]. Each marker indicates a trackers accuracy and speed measured in frames per second (FPS) [Hz], i.e., higher and more right is better

In the paper A. Bewley et al used the Faster Region CNN (Faster RCNN) detection framework [60]. Faster RCNN is an end-to-end framework that consists of two stages. The first stage extracts feature and proposes regions for the second stage which then classifies the object in the proposed region. (Figure 2.6) The advantage of this framework is that parameters are shared between the two stages creating an efficient framework for detection. Additionally, the network architecture itself can be swapped to any design which enables rapid experimentation of different architectures to improve the detection performance. The FrRCNN was used with default parameters learnt from PASCAL VOC challenge dataset as the work was only interested in pedestrians. like the purpose of this master thesis.

2.2.2.1.2– Estimation Model

the object model, i.e., the representation and the motion model used to propagate a target's identity into the next frame. by approximating the inter-frame displacements of each object with a linear constant velocity model which is independent of other objects and camera motion.

The state of each target is modelled as:

$$X = [\, u, v, s, r, u, \dot{u}, \dot{v}, \dot{s}]^{T}$$

Where u and v represent the horizontal and vertical pixel location of the centre of the target (the center of the bounding box), while the scale s and r represent the scale (area) and the aspect ratio of the target's bounding box respectively. the aspect ratio as mentioned in the paper [58] is considered to be constant. When a detection is associated to a target, the detected bounding box is used to update the target state where the velocity components are solved optimally via a Kalman filter framework [60]. If no detection is associated to the target, its state is simply predicted without correction using the linear velocity model.

2.2.2.1.3 – Data Association

In assigning detections to already existing targets, each target's bounding box geometry is estimated by predicting its new location in the current frame. The assignment cost matrix is then computed as the intersection-over-union (IOU) (section reference) distance between each detection and all predicted bounding boxes from the existing targets. The assignment is solved optimally using the Hungarian algorithm. Additionally, a minimum IOU is imposed to reject assignments where the detection to target overlap is less than minimum IOU.

2.2.2.1.4 – Occlusion handling

In terms of occlusion handling A. Bewley et al [58], found that the IOU distance of the bounding boxes implicitly handles short term occlusion caused by passing targets. Specifically, when a target is covered by an occluding object, only the occluding object is detected, since the IOU distance appropriately favours detections with similar scale. This allows both the occluder target to be corrected with the detection while the covered target is unaffected as no assignment is made because it's not detected in the frame.

2.2.2.1.5 - Creation and Deletion of Track Identities

When objects enter and leave the image, unique identities need to be created or destroyed accordingly. For creating trackers, we consider any detection with an overlap less than IOUmin to signify the existence of an untracked object. The tracker is initialised using the geometry of the bounding box with the velocity set to zero. Since the velocity is unobserved at this point the covariance of the velocity component is initialised with large values, reflecting this uncertainty. Additionally, the new tracker then undergoes a probationary period where the target needs to be associated with detections to accumulate enough evidence in order to prevent tracking of false positives. Tracks are terminated if they are not detected for TLost frames (e.g., if the target is not detected in 30 frames, we consider the target either left the frame or a lost track). This prevents an unbounded growth in the number of trackers and localisation errors caused by predictions over long durations without corrections from the detector.

While achieving overall good performance in terms of tracking precision and accuracy, SORT in fact returns a high number of identity switches (IDSW) (*see chapter 1, Object tracking metrics*). This is, because the employed association metric is only accurate when state estimation uncertainty is low. Therefore, SORT has a deficiency in tracking through occlusions. overcoming this issue, by replacing the association metric with a more informed metric that combines motion and appearance information. More precisely, they apply a convolutional neural network (CNN) that has been trained to discriminate pedestrians on a large-scale person re-identification dataset. Through integration of this network, to increase robustness against misses and occlusions while keeping the system easy to implement, efficient, and appliable to online scenarios, and it's called DeepSORT.

2.2.2.2– Simple Online Realtime tracker with Deep associating metric (DeepSORT)

The Simple Online and Realtime Tracking with a Deep Association metric (Deep SORT) enables multiple objects tracking by integrating appearance information with its tracking components [58]. A combination of Kalman Filter [62] and Hungarian algorithm [63] is used for tracking. Here, Kalman filtering is performed in image space while Hungarian technique facilitates frame-by-frame data association using an association metric that computes bounding box overlap. To obtain motion and appearance information, a trained convolutional neural network (CNN) is applied.

2.2.2.2.1– Track managing & Estimation Model Update

The track handling and Kalman filtering framework is mostly identical to the original formulation in SORT [62], the tracking scenario is defined on the eight-dimensional state space $X = (u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h})$.

Where the bounding box center position (u, v), aspect ratio γ, height h, and their respective velocities in image coordinates. they used a standard Kalman filter with constant velocity motion and linear observation model.

For each track k they count the number of frames since the last successful measurement association C_k. This counter is incremented during Kalman filter prediction and reset to 0 when the track has been associated with a measurement (detection). Tracks that exceed a predefined maximum age A_{max} are considered to have left the scene and are deleted from the track set. New track hypotheses are initiated foreach detection that cannot be associated to an existing track. These new tracks are classified as tentative during their first three frames. During this time, we expect a successful measurement association at each time step. Tracks that are not successfully associated to a measurement within their first three frames are deleted (explained more in the *Implementation chapter 3*).

2.2.2.2.2 – Motion Estimation

Calculate the Mahalanobis distance (the distance between two points in multivariate space) between the position predicted by Kalman Filter and the newly detected object position

$$d^{(1)}(i,j) = (\boldsymbol{d}_j - \boldsymbol{y}_i)^{\mathrm{T}} \boldsymbol{S}_i^{-1} (\boldsymbol{d}_j - \boldsymbol{y}_i), \tag{2.2}$$

where projection of the i-th track distribution into measurement space by (***yi***, ***Si***) and the j-th bounding box detection by ***dj***. The Mahalanobis distance takes state estimation uncertainty into account by measuring how many standard deviations the detection is away from the mean track location. If the Mahalanobis distance of an association is less than the specified threshold the association of the set motion state is successful.

$$b_{i,j}^{(1)} = \mathbb{1}[d^{(1)}(i,j) \leq t^{(1)}] \tag{2.3}$$

Where the corresponding Mahalanobis threshold denoted in the paper is $t^{(1)}$ = 9.4877

2.2.2.2.3 – Appearance features

When the uncertainty of motion is very low, the above-mentioned Mahalanobis matching is a suitable method of correlation measurement, but the use of Kalman filtering to estimate motion state in image space is only a rough prediction. Especially when the camera is in motion, the correlation method of Mahalanobis distance will be invalid, resulting in the phenomenon of ID switch. Therefore, appearance features are introduced. The minimum value of the cosine distance between the current detection's feature vector (requires || r || = 1) and each associated feature vector in tracks is calculated.

$$d^{(2)}(i,j) = \min\{1 - \boldsymbol{r}_j^{\mathrm{T}} \boldsymbol{r}_k^{(i)} \mid \boldsymbol{r}_k^{(i)} \in \mathcal{R}_i\}. \tag{2.4}$$

where $\boldsymbol{r}_j$ is the appearance descriptor extracted from within the j^{th} detected bounding box, and R_i is the set of the last 100 appearance descriptors r_k^i associated with the i^{th} track.

The $d^{(2)}$ measure uses the cosine distance between the j^{th} detection and a number of detections already assigned to i^{th} track, so if a visually similar detection was previously seen, the distance will be low. If the above distance is less than the specified threshold, then the visual association is considered successful.

The final measurement matrix:

$$c_{i,j} = \lambda\, d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{2.5}$$

Where we call an association admissible if it is within the gating region of both metrics:

$$b_{i,j} = \prod_{m=1}^{2} b_{i,j}^{(m)}. \tag{2.6}$$

Only when $c_{i,j}$ is within the intersection of the two metric thresholds, it is considered that the correct association is achieved. The distance metric is good for short-term prediction and matching, but for long-term occlusion, it is more effective to use the appearance feature metric. For the case of camera movement, you can set $\lambda = 0$, where we only consider the cosine distance (feature vector). To be: $c_{i,j} = d^{(2)}(i,j)$

2.2.2.2.4– Cascade matching

When a target is occluded for a long time, the uncertainty of Kalman filter prediction with the object location will greatly increase, and the observability in the state space will be greatly reduced. **If at this time two trackers compete for matching the same detection resulted**, the Mahalanobis distance favours larger uncertainty, because it effectively reduces the distance in standard deviations of any detection towards the projected track average. This is an undesired behaviour as it can lead to increased track fragmentations (*see chapter 1, MOT metrics*) and unstable tracks. Therefore, they introduced the matching cascade, where the matching cascading ensures to give priority of the recent tracks.

Listing 1 Matching Cascade

Input: Track indices $\mathcal{T} = \{1, \ldots, N\}$, Detection indices $\mathcal{D} = \{1, \ldots, M\}$, Maximum age $A_{\max}$

Compute cost matrix $\boldsymbol{C} = [c_{i,j}]$ using Eq. 5
Compute gate matrix $\boldsymbol{B} = [b_{i,j}]$ using Eq. 6
Initialize set of matches $\mathcal{M} \leftarrow \emptyset$
Initialize set of unmatched detections $\mathcal{U} \leftarrow \mathcal{D}$
for $n \in \{1, \ldots, A_{\max}\}$ **do**
    Select tracks by age $\mathcal{T}_n \leftarrow \{i \in \mathcal{T} \mid a_i = n\}$
    $[x_{i,j}] \leftarrow \text{min\_cost\_matching}(\boldsymbol{C}, \mathcal{T}_n, \mathcal{U})$
    $\mathcal{M} \leftarrow \mathcal{M} \cup \{(i,j) \mid b_{i,j} \cdot x_{i,j} > 0\}$
    $\mathcal{U} \leftarrow \mathcal{U} \setminus \{j \mid \sum_i b_{i,j} \cdot x_{i,j} > 0\}$
end for
return $\mathcal{M}, \mathcal{U}$

Figure 2.18 matching algorithm pseudo code from [59]

As input track T and detection D indices as well as the maximum age A_{max}. In lines 1 and 2 compute the association cost matrix and the matrix of admissible associations (Eq 2.5, Eq 2.6). then iterate over track age n to solve a linear assignment problem for tracks of increasing age. In line 6 select the subset of tracks T_n that have not been associated with a detection in the last n frames. In line 7 solve the linear assignment between tracks in $T_n$and unmatched detections U.

In a final matching stage, run intersection over union association as proposed in the original SORT algorithm [58] on the set of unconfirmed and unmatched tracks of age n = 1. This helps to account for sudden appearance changes, e.g., due to partial occlusion with static scene geometry, and to increase robustness against erroneous initialization.

2.2.2.2.5 - Deep Appearance Descriptor

By integrating CNN, the tracker achieves greater robustness against object misses and occlusions while preserving the trackers' ability to quickly implement to online and Realtime scenarios. The CNN architecture of the system is shown in Table 2.3. A wide residual network with two convolutional layers followed by six residual blocks is applied. In dense layer 10, a global feature map of dimensionality 128 is calculated. Finally, batch and ℓ2 normalization features over the unit hypersphere accesses compatibility with cosine arrival metric. Overall, DeepSORT is a highly versatile tracker and can match performance capabilities with other state-of-the-art tracking algorithms.

Name	Patch Size/Stride	Output Size
Conv1	3 × 3/1	32 × 128 × 64
Conv2	3 × 3/1	32 × 128 × 64
Max Pool 3	3 × 3/2	32 × 64 × 32
Residual 4	3 × 3/1	32 × 64 × 32
Residual 5	3 × 3/1	32 × 64 × 32
Residual 6	3 × 3/2	64 × 32 × 16
Residual 7	3 × 3/1	64 × 32 × 16
Residual 8	3 × 3/2	128 × 16 × 8
Residual 9	3 × 3/1	128 × 16 × 8
Dense 10		128
Batch and ℓ_2 normalization		128

Table 2.3 DeepSORT Re-Identification network [67]

Successful application of DeepSORT method requires a well-discriminating feature embedding to be trained offline, before the actual online tracking application. To this end, they employed a CNN (table 2.3). The network was pre-trained on a large-scale person re-identification dataset [90] of more than a million images of 1261 pedestrians. The appearance information helps with re-identification of objects that have not been tracked for longer time because of missed detections, and or because they were under occlusion or because they have briefly left the scene.

SORT and DeepSORT work flow will be explained in greater more details in the implementation chapter

2.3 - Object Detection and Tracking Algorithms for Vehicle Counting

In November 2020, the authors Vishal Mandal and Yaw Adu-Gyamfi [67] deployed several state-of-the-art object detection and tracking algorithms to detect and track different classes of vehicles in their regions of interest (ROI). The goal of correctly detecting and tracking vehicles in their ROI is to obtain an accurate vehicle count. Multiple combinations of object detection models coupled with different tracking systems are applied to access the best vehicle counting framework. The models' addresses challenges associated to different weather conditions, occlusion and low-light settings and efficiently extracts vehicle information and trajectories through its computationally rich training and feedback cycles. The automatic vehicle counts resulting from all the model combinations are validated and compared against the manually counted ground truths of over 9 h' traffic video data obtained from the Louisiana Department of Transportation and Development

2.3.1 Methodology

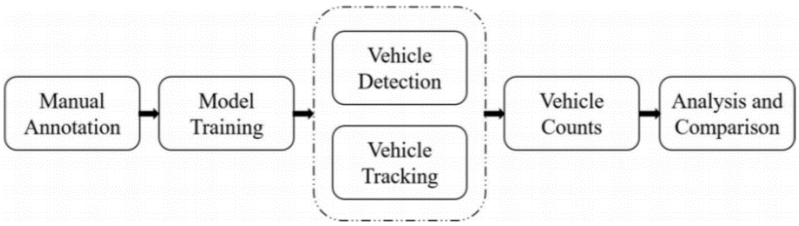


Figure 2.19 Detection based tracking vehicle counting framework

The authors proposed vehicle counting framework (Figure 2.20) initiates by manually annotating traffic images. This is followed by training several object detections models which can then be used to detect different classes of vehicles. After obtaining detection results for each video frame, different tracking algorithms are used for multi-object tracking. In this study, they used both online and offline tracking algorithms. Although offline tracking algorithms yield better results, but it won't realize in applications that involve online traffic control scenarios. The green and blue polygons drawn on the cameras (see Fig 2.8) assigns the entrance and exit zones for every vehicle's trajectory and computes the number of vehicles passing through the north and southbound directions, respectively

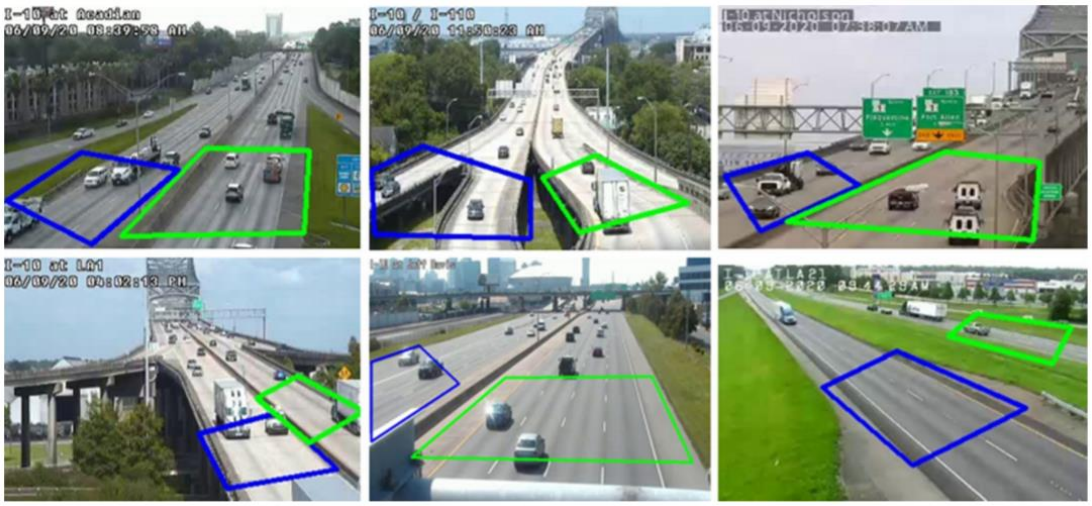


Figure 2.21 Camera locations for the comparative study [67]

2.3.2- Used Algorithms for the comparative study

Four different state-of-the-art object detectors and trackers were used making a total of 16 different detector-tracker combinations. Upon obtaining vehicle counts, all these detector-tracker combinations were further analysed and had their performance capabilities compared based of different environmental conditions. The Object detectors and trackers that were used in this analysis studies were as follows:

- Object Detectors:

    a. YOLOv4 [75]
    b. EfficientDet [70]
    c. CentreNet [68]
    d. Detectron2 [69]

- Object Trackers:

    a. SORT [58]
    b. DeepSORT [59]
    c. IOU Tracker [71]
    d. Kalman IOU (KIOU) [72]

2.3.3-Results

They evaluated the performance of different combinations of object detectors and trackers. The main goal of their study is to identify the best performing object detector tracker combination. For comparative analysis, the models are tested on a total of 546 video clips of length 1 min each comprising of over 9 h' total video length. Figure 2.21 shows all the camera views with manually generated green and blue polygons that record the number of vehicles passing through them in both north and southbound directions, respectively. The vehicle counts are evaluated based on four different categories: (1) overall count of all vehicles, (2) total count of cars only, (3) total count of trucks only,

and (4) overall vehicle counts for different times of the day (i.e., daylight, night-time, rain). To establish ground truth, all the vehicles are manually counted from the existing 9 h' video test data.

- **Cars Counting Results:**

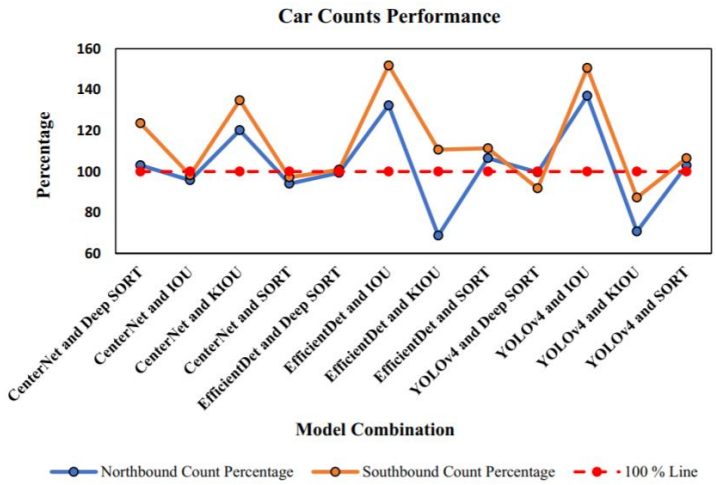


Figure 2.22 Performance of model combination for car counts only [67]

- **Truck counting Results:**

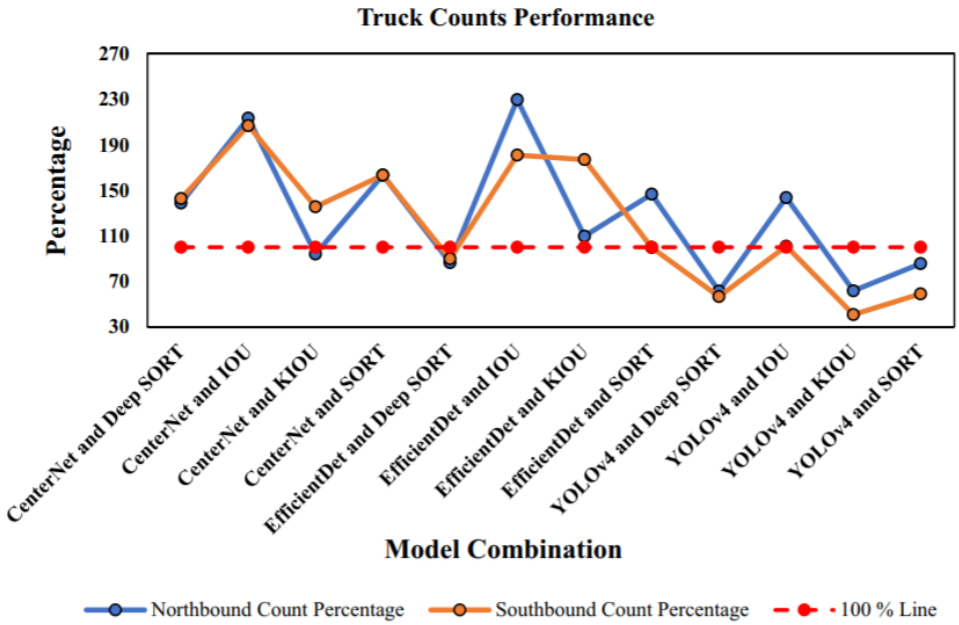


Figure 2.23 Performance of model combination for car counts only [67]

- **Results Conclusion:**

Occlusion and lower visibility created identity switches and same vehicles were detected multiple times which caused the model to sometimes over-exaggerate the number of vehicles. Although, conditions such as inferior camera quality, occlusion and low light conditions proved tricky in

accurately detecting different classes of vehicles, Overall, for counting all vehicles on the roadway, experimental results from this study proved that there's two powerful combination of object Detectors and trackers are YOLOv4 and Deep SORT, and CenterNet and Deep SORT were the most ideal combinations. In the table below shows the count percentages of Northbound (NB) and Southbound (SB), the green and blue polygons (in Figure 2.21) respectively.

Combination	Daylight (NB, SB)	Rain (NB, SB)
YOLOv4 + DeepSORT	(92.27%, 91.58%)	(91.25%, 89.25%)
CenterNet + DeepSORT	(97.42%, 100.13%)	(102.00%, 87.23%)

Table 2.4 Comparison of (Yolov4+DeepSORT) and (CentreNet+DeepSORT) in percentage of counting accuracy [67]

Time of day	Model combination	Northbound count percentage	Southbound count percentage
Daylight	YOLOv4 and SORT	112.3597165	114.9770576
	YOLOv4 and KIOU	70.81364442	89.70461715
	YOLOv4 and IOU	144.3812758	155.2767422
	YOLOv4 and Deep SORT	92.277562	91.5865623
	EfficientDet and SORT	30.53610906	23.24962286
	EfficientDet and KIOU	32.47185667	41.27978954
	EfficientDet and IOU	82.04832193	57.673148
	CenterNet and SORT	114.2941524	115.5147691
	CenterNet and KIOU	75.02215003	105.6638945
	CenterNet and IOU	137.0496161	144.063665
	CenterNet and Deep SORT	97.42321323	100.1362202
Night-time	YOLOv4 and SORT	107.1243523	106.5976714
	YOLOv4 and KIOU	72.99222798	87.12807245
	YOLOv4 and IOU	145.9196891	166.2354463
	YOLOv4 and Deep IOU	91.256962	90.25664125
	EfficientDet and SORT	59.45595855	55.62742561
	EfficientDet and KIOU	36.1952862	36.01368691
	EfficientDet and IOU	76.52011225	53.14617619
	CenterNet and SORT	110.880829	107.7619664
	CenterNet and KIOU	74.74093264	112.4191462
	CenterNet and IOU	144.753886	161.3842173
	CenterNet and Deep SORT	95.98445596	92.94954722
Rain	YOLOv4 and SORT	114.4578313	101.9874477
	YOLOv4 and KIOU	82.06157965	74.89539749
	YOLOv4 and IOU	145.9170013	153.7656904
	YOLOv4 and Deep SORT	91.258975	89.256987
	EfficientDet and SORT	46.18473896	47.90794979
	EfficientDet and KIOU	49.45567652	46.2195122
	EfficientDet and IOU	92.32	55.47169811
	CenterNet and SORT	131.8607764	107.1129707
	CenterNet and KIOU	119.1432396	99.47698745
	CenterNet and IOU	169.7456493	150.3138075
	CenterNet and Deep SORT	102.0080321	87.23849372

Figure 2.24 comparison table from the paper [67]

2.4- Proposed Framework

- **In terms of Accuracy**

As indicated in the SORT paper [58], detection quality is identified as a key factor influencing tracking performance, Vishal Mandal and Yaw Adu-Gyamfi analysis study [67], both combinations included the *State-of-Art* DeepSORT algorithm, we can see also that the two detectors (YOLOv4, CenterNet) didn't perform that well with the other Detection-Based-Trackers (SORT, KIOU, IOU). As shown in the tables below (See table 2.5, 2.6, 2.7), that during daylight and night time (low lighting) and even in the rainy weather which means in a more challenging conditions (more occlusion to the detector and the tracker), the DeepSORT algorithm stayed at top performance regardless of the detector in terms of accuracy.

Note: the closer the combination gets to 100%, the better. If the percentage is above 100% that means the combination counted more vehicles than the ground truth. (More count than the actual vehicles passed)

NB: Northbound count percentage (Green Polygon).

SB: Southbound count percentage (blue Polygon) (see Figure 2.21).

- ***Day light:***

	YOLOV4 (NB, SB)	**CENTRENET (NB, SB)**	**EFFICIENTDET (NB, SB)**
SORT	(112.35, 114.97)	(114.29, 115.51)	(30.53, 23.24)
DEEPSORT	(**92.27, 91.58**)	(**97.42, 100.13**)	/
KIOU	(70.81, 89.70)	(75.02, 105.66)	(32.47, 41.27)
IOU	(144.38, 155.27)	(137.04, 144.06)	(82.04, 57.67)

Table 2.5 comparison of the top performing combination in daylight [67]

- ***Night-time:***

	YOLOV4 (NB, SB)	**CENTERNET (NB, SB)**	**EFFICIENTDET (NB, SB)**
SORT	(107.12, 106.59)	(110.88, 107.76)	(59.45, 55.62)
DEEPSORT	(**91.25**, **90.25**)	(**97.42**, **100.13**)	/
KIOU	(72.99, 87.12)	(74.74, 112.41)	(36.19, 36.01)
IOU	(145.91, 166.23)	(144.75, 161.38)	(76.52, 53.14)

Table 2.6 comparison of the top performing combination in Night-Time [67]

- ***During Rain(occlusion)*:**

	YOLOV4 (NB, SB)	CENTERNET (NB, SB)	EFFICIENTDET (NB, SB)
SORT	(114.45, 101.98)	(131.86, 107.11)	(46.18, 47.90)
DEEPSORT	(**91.25, 89.25**)	(**102.00, 87.23**)	/
KIOU	(82.06, 74.89)	(119.14, 99.47)	(49.45, 46.21)
IOU	(145.91, 153.76)	(169.74, 150.31)	(92.32, 55.47)

Table 2.7 comparison of the top performing combination during Rain [67]

By looking to these tables our choice on the tracking algorithm couldn't be easier to build our Pedestrian tracking module. We can see also that CenterNet & DeepSORT performed slightly better than YOLOv4 & DeepSORT.

- **In terms of Speed**

In terms of speed which is an important factor in this project (Realtime for a Live video scenario), we observe that the yolov4 FPS outperform the CenterNet FPS by a large margin with a comparable mAP score in COCO dataset, a good trade-off between accuracy & speed [73].

Note: the smaller the *interference time* (*MS*), means faster the training

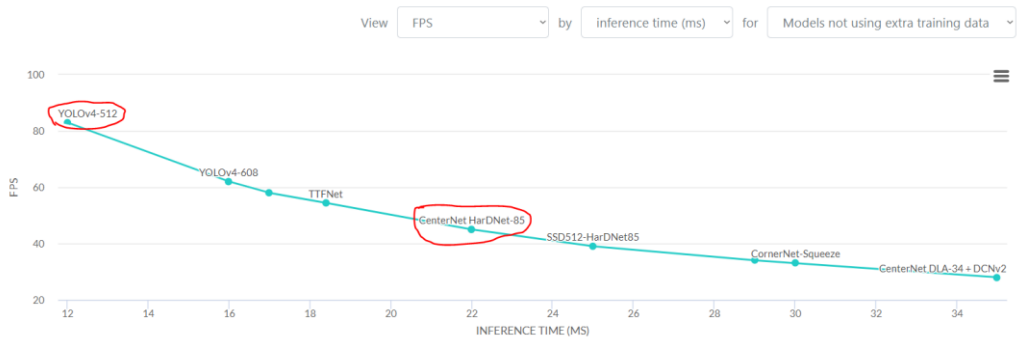


Figure 2.25 Real-Time Object Detection on COCO [73]

YOLOv4 [75] neural network and the Darknet Deep Learning framework (C/C++/CUDA) are better in FPS speed and AP50:95 and AP50 accuracy, on Microsoft COCO dataset [74], than the other DL-frameworks and neural networks (Google TensorFlow EfficientDet, Facebook Detectron RetinaNet/MaskRCNN, PyTorch Yolov3-ASFF, and many others …). YOLOv4 achieves 43.5% AP / 65.7% AP50 accuracy according to Microsoft COCO test at speed **62 FPS** TitanV or 34 FPS RTX 2070. Unlike other modern detectors such as CenterNet, YOLOv4 can be trained by anyone who uses the Nvidia gaming graphics adapter with 8–16 GB VRAM. Now, not only large companies can train a neural network on dozens of GPUs / TPUs using large mini-batch sizes to achieve higher accuracy.

YOLOV4 is optimal for real-time object detection tasks because the network lies on the Pareto optimality curve of the AP (accuracy) / FPS (speed) chart:

Note:

- ❖ CenterNet article [68] stated their FPS only on Pascal GPU (see red circles and arrows in Figure 2.26)
- ❖ AP50 means Average Precision over 50%. (See **Chapter 1, 2.2.1- object detection metrics**)

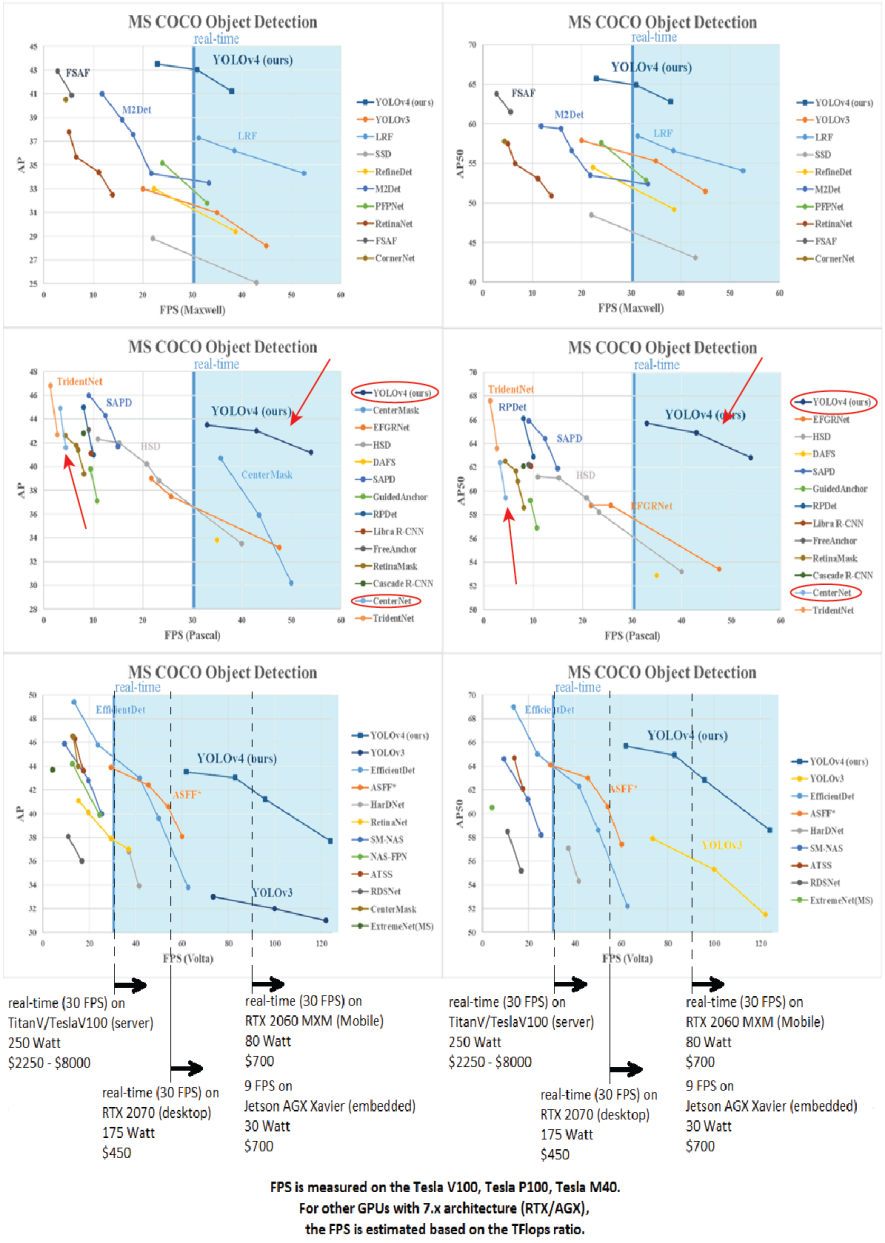


Figure 2.26. (modified) Comparison of the speed and accuracy of different object detectors. (Some articles stated the FPS of their detectors for only one of the GPUs: Maxwell/Pascal/Volta), [75]

Comparison of the results obtained with other state of-the-art object detectors are shown in Figure 2.26. more specifically with CenterNet because it showed excellent compatibility with the DeepSORT algorithm. The YOLOv4 are located on the Pareto optimality curve and are superior to the fastest and most accurate detectors in terms of both speed and accuracy.

We can conclude that in terms of speed the YOLOv4 is way faster than CenterNet and slightly better in AP50 metric (see Figure 2.26). it is a fair and reasonable trade-off between accuracy and speed to choose the Yolov4 as a detector when the *real-time* is of one the major factors of a given project.

Conclusion

In order to build a robust multi pedestrian application that balances between accuracy and speed for a real time application.

The First thing is to choose the right processing mode, which is the "*online tracking"* that only relays on the Current frame and previous frame only without peeking on future frames, to serve the real-time purpose and be appliable to real-life **Live** scenarios (such as live football match, security camera, autonomous vehicles, …etc), and we even can-do tracking for pre-recorded videos, "*offline tracking*" with an "*Online* tracking" processing fashion.

We saw that DeepSORT is the best fit *online-tracker* and *Detection-Based tracker* (explained in **1.3-MOT Categorization**) for this master thesis project. But unlike as mentioned in the SORT with deep association metric (DeepSORT) paperwork [59] where they used the Faster-RCNN [54] as a detector. We are going to replace the Faster-RCNN with a better and faster detector YOLOv4 that performs incredibly well with DeepSORT, and since we know the detection quality is the key factor for better tracking performance as mentioned in SORT paperwork [58] where changing the detector lead up to 18.9% in the MOTA score (explained in **chapter 1, 2.2.2 – Object tracking metrics**) on). And duo to the flexibility of the YOLOv4 architecture the model can be retrained to become a multi object detector with a binary classifier ("Pedestrian_YOLOv4" explained in details in **Implementation chapter**) where the model will only be focusing on detecting pedestrians' locations, after reimplementing the "pedestrian_YOLOv4 (ours)" model to a TensorFlow model to be able to do GPU computation for faster calculation during the video to feed the DeepSORT algorithm (which will also be reimplemented and explained in details in the **Implementation chapter**), for a better and faster pedestrian tracking performance.

- **Adopted Framework for Multi-Object (Pedestrian) Tracking in Realtime:**

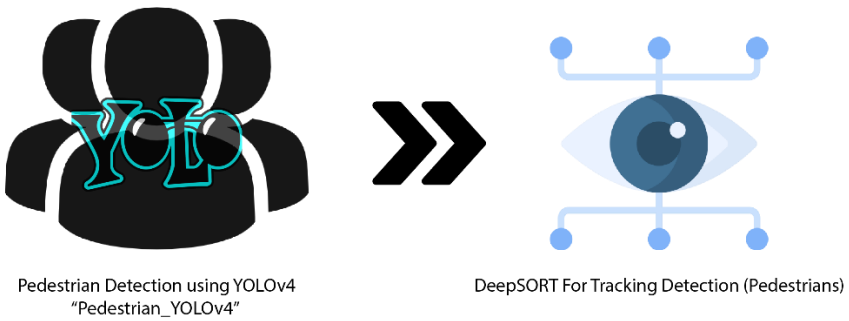


Figure 2.27 optimal speed vs accuracy proposed Framework

Chapiter 3: Project Design

Introduction

The problem of Multiple Object Tracking (MOT) consists in following the trajectory of different objects in a sequence, usually a video. In recent years, with the rise of Deep Learning, this field of study has attracted many researchers, and many algorithms have been created to provide a solution to this problem, those algorithms have benefited from the representational power of deep models.

In this work we are going to build a robust module that can detect and track pedestrians with good accuracy & speed to preserve the Realtime time challenge and in a scenario where we don't know the future frames (Online tracking method e.g., Live webcams, Live football game…etc.) and it can also run with pre-recorded video in an online manner where it depends only in the current frame and the previous one (no looking to the future)

3.1- System Design

3.1.1 Methodology

In our system, as seen in chapiter 2 in the DeepSORT paperwork they have used the Faster RCNN in the detection phase, so we are going to replace FrRCNN [54] with the an implemented YOLOv4 architecture (Pedestrian_YOLOv4 that focuses on pedestrians with a personal custom dataset), and running the model in a GPU environment so the Object (pedestrian) detection phase will be quicker and Realtime wise, then reimplanting the state-of-the-art algorithm DeepSORT for outputting tracks and test it in a real world scenario.

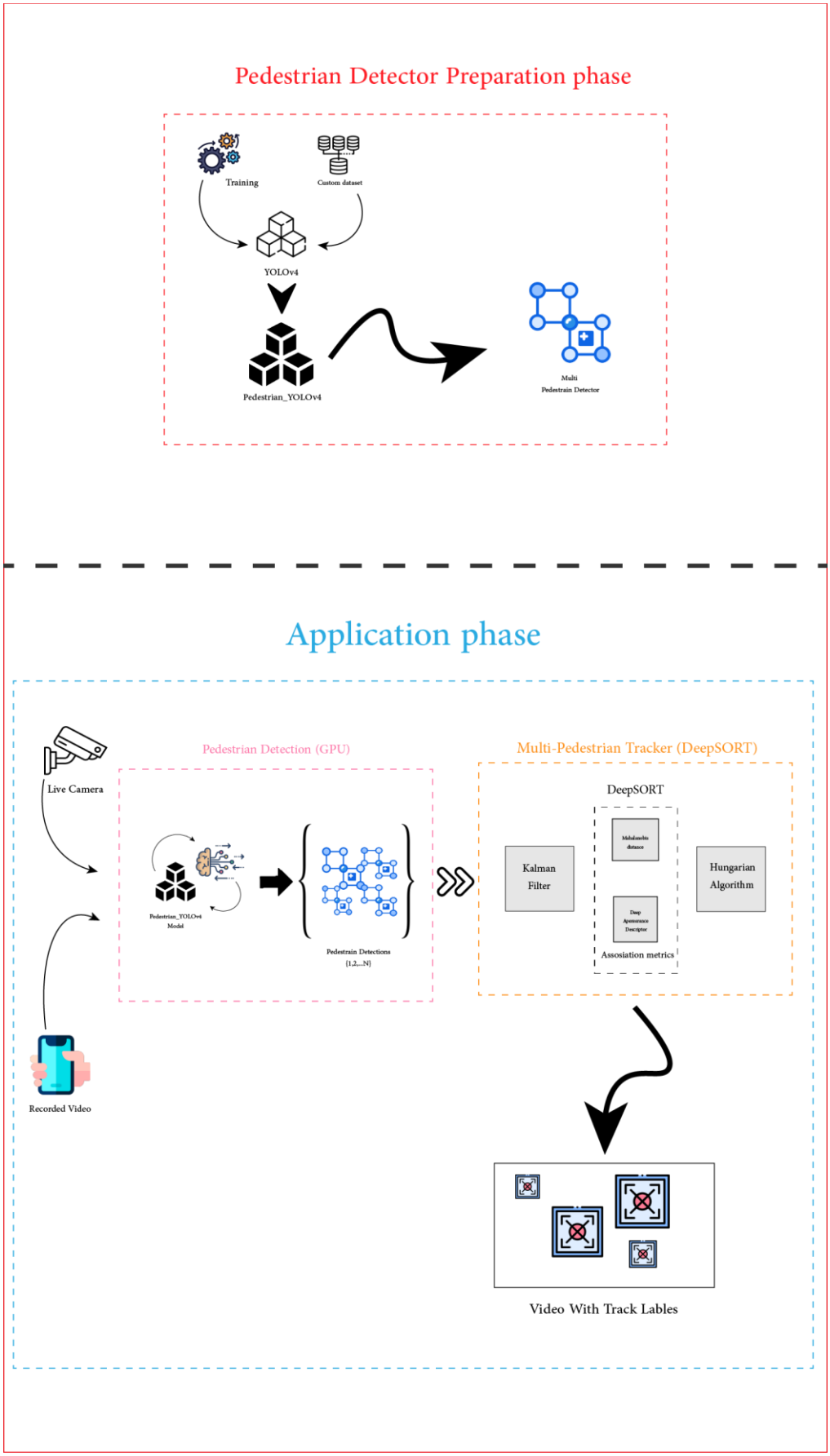


Figure 3.1 Global design

3.1.2 – Detailed System Design

3.1.2.1 – Pedestrian Detector Preparation Phase

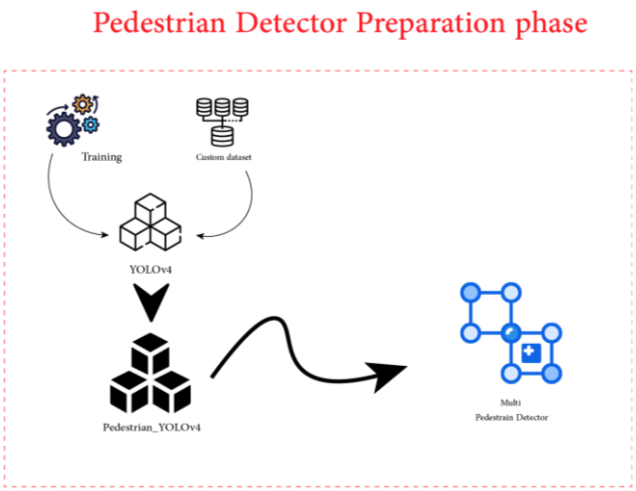


Figure 3.2 Pedestrian Detector Preparation phase

Object detection models continue to get better, increasing in both performance and speed. In the Realtime object detection space, YOLOv3 [76] (released April 8, 2018) has been a popular choice, as has EfficientDet [70] (released April 3rd, 2020) by the Google Brain team. Progress continues with the recent release of YOLOv4 [75] (released April 23rd, 2020), which has been shown to be the new object detection champion by standard metrics on COCO .(see chapiter 1, Object Detection Datasets)

These general object detection models are proven out on the COCO dataset which contains a wide range of objects and classes with the idea that if they can perform well on that task, they will generalize well to new datasets. However, applying the deep learning techniques used in research can be difficult in practice on custom objects.

Duo to the flexibility of YOLOv4 architecture the model can be trained on multiple Object classes (Dogs, Cats, ... Etc), As in our case we are going to make the Architecture focus on Pedestrian tracking, what that means is we are only interested in detecting pedestrians.

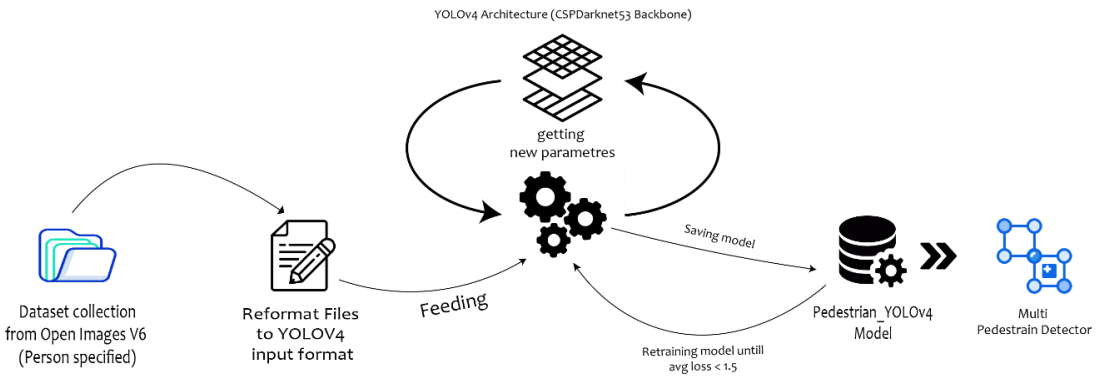


Figure 3.3 – Setting the Pedestrian_YOLOv4

3.1.2.1.1– Dataset Collection

The best way to improve an object detection model is to gather more representative data, and YOLO v4 does not escape from this fact either. As Tesla's Senior Director of AI, Andrej Karpathy puts it when explaining how Tesla teaches cars to stop in Conference on Computer Vision and Pattern Recognition in 2020 [99].

- **Open Images Dataset V6:** Open Images is a dataset of ~9M images annotated with image-level labels, object bounding boxes, object segmentation masks, visual relationships, and localized narratives: It contains a total of 16 million bounding boxes for 600 object classes on 1.9M images, making it the *largest existing dataset with object location annotations*. The boxes have been largely manually drawn by professional annotators to ensure accuracy and consistency. The images are very diverse and often contain complex scenes with several objects (8.3 per image on average).

  As we are mainly interested in images that contains pedestrians with their bounding boxes coordinate. To download all the training and validation one by one it would be time consuming. so, thanks to Vittorio Mazzia [link to github] for creating a toolkit that helps to make the process easier by creating the *OIDv4_ToolKit* where we can download the dataset with Python scripts.

3.1.2.1.2- Dataset Reformatting

A. YOLOv4 Data Annotation Input

YOLOv4 input format is `.txt`-**file** for each `.jpg`-**image-file** - in the same Folder and with the same **name**.
in the `.txt`-**extension**, put to file: object class (which is a number represents the class name) and object coordinates on this image and separation each information with the space.
For each object in new line in the .txt-extension:

```
<object-class> <x> <y> <width> <height>
```

Note: <x>, <y> here are the center of the bounding box and width and height are scaled to the image's shape

Figure 3.4 example from our Dataset

B. Dataset Annotation Reformatting

In our case we split the dataset in **4000** images for training and **400** images for validation, the *OIDv4_ToolKit* gives a folder contains two folders

a. **CSV folder**: Train annotation and validation annotation and a class descriptor with **.csv-extension**
b. **Dataset folder**: Containing the training and validation folder
    i. **Training and validation folder**: Contains a Label folder and images for each folder (1500 images in training folder and 400 for validation dataset)
    ii. **Label folder: .txt** files that correspond to each image in training or validation folder Annotated as**:**
    <Class_name (string**)**> <Xmin><Xmax><Ymin><Ymax>

As we are just interested the training and validation folder, we either reformate these files manually by creating **.txt** files for each image (very time consuming) or create a python script that do the work for us.

```
# function that turns XMin, YMin, XMax, YMax coordinates to normalized yolo format
def convert(filename_str, coords):
    os.chdir("..")
    image = cv2.imread(filename_str + ".jpg")
    coords[2] -= coords[0]
    coords[3] -= coords[1]
    x_diff = int(coords[2]/2)
    y_diff = int(coords[3]/2)
    coords[0] = coords[0]+x_diff
    coords[1] = coords[1]+y_diff
    coords[0] /= int(image.shape[1])
    coords[1] /= int(image.shape[0])
    coords[2] /= int(image.shape[1])
    coords[3] /= int(image.shape[0])
    os.chdir("Label")
    return coords

ROOT_DIR = os.getcwd()
```

Figure 3.5 python script that converts annotation

3.1.2.1.3– Training **Pedestrian_YOLOv4**

For this purpose, we are going to use the Darknet framework [92], in Google Collab instead of Pytorch or TensorFlow for parallel training (Google Collab offers free powerful NVIDIA Tesla K80 GPU for Network training)

- **Darknet Framework:** Darknet is an open-source neural network framework [92] written in C language and CUDA. It is fast easy to install and supports CPU and GPU computation. Darknet is installed with only two computational dependencies: OpenCV if users want to use a wider variety of supported image types or CUDA if we want GPU computation like in our case.

YOLOv4 is a general object detection model which has proven out on the COCO [21] dataset which contains a wide range of objects and classes with the idea that if they can perform well on that task, they will generalize well to new datasets. By using YOLOv4 we are implementing many of the past research contributions in the YOLO family alongside with new Data Augmentation techniques (*we will be discussed later*).

Since YOLOv4 model already been trained on the COCO, ImageNet, Crowd human datasets, we want to make YOLOv4 only focuses on detecting Pedestrians, as we already have the correct annotation for training and validation for YOLOv4, and also, we don't have to train the network from scratch so we will use the ***Transfer learning approach***.

- **Transfer Learning approach:** we take a model trained on a large dataset such as COCO dataset and ImageNet and transfer its knowledge to a smaller dataset. For object detection and recognition with a CNN, we freeze the early convolutional layers of the network and only train the last few layers which make a prediction. The idea is the convolutional layers extract general, low-level features that are applicable across images (such as edges, patterns, gradients, …etc), and the later layers identify specific features within an image. Thus, we can use a network trained on unrelated categories in a massive dataset and apply it to our own problem because there are universal, low-level features shared between images. The images in the Open Images V6 [23] dataset are very similar to those in the COCO & ImageNet datasets and the knowledge a model learns on those datasets will easily transfer to this task.

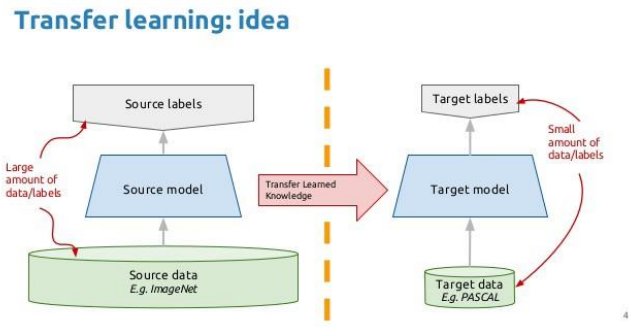


Figure 3.6 Transfer learning approach [65]

- **Model Variables changes:**

Items	YOLOv4	Pedestrian_YOLOv4
Classes	80	1 (pedestrian)
Classification	Multiple	Binary
Input size (Width, Height)	(512,512)	(416,416)
Filter (Last Convolution layer of CSPDarknet 53)	255	18
Max_batches	500500	6000
Steps	400000,450000	4800, 5400

Table 3.1 Model Variables changes

3.1.2.1.4 – Data Augmentation

By using YOLOv4, we are implementing many of the past research contributions in the YOLO family along with a series of new contributions unique to YOLOv4 [75]. In order to make the designed detector more suitable for training on single GPU, they made additional design and improvement as follows:

A. Self-Adversarial Training (SAT):

This technique uses the state of the model to inform vulnerabilities by transforming the input image [75]. First, the image is passed through a normal training step. Then, rather than back-propping through the weights, the loss signal is used to alter the image in a way that would be most detrimental to the model. Later on in training, the model is forced to confront this particularly hard example and learn around it.

B. Mosaic data augmentation:

the Mosaic augmentation was invented by Glenn Jocher earlier last year and was first released in YOLO v4 [75]. It has made quite a splash. It works by taking four source images and combining them together into one. This does a few things:

Figure 3.7 Example mosaic output [75].

- It simulates four **random crops** (while maintaining the relative scale of the objects compared to the image) which can help the model perform better in cases of occlusion and translation. which we really need in the tracking process
- It **combines classes** that may not be seen together in your training set (for example, if we have pictures of apples and pictures of oranges, but no pictures of apples with oranges in the same photo, mosaic will simulate that).
- It varies the number of objects in the images (for example, if all of our images only contain one **bounding box**, the output of mosaic will have between zero and four).

C. CutOut data augmentation:

another advanced augmentation is CutOut also first seen in the Yolov4, Cutout simulates occlusion by adding randomly generated black boxes on top of the images. This does two things

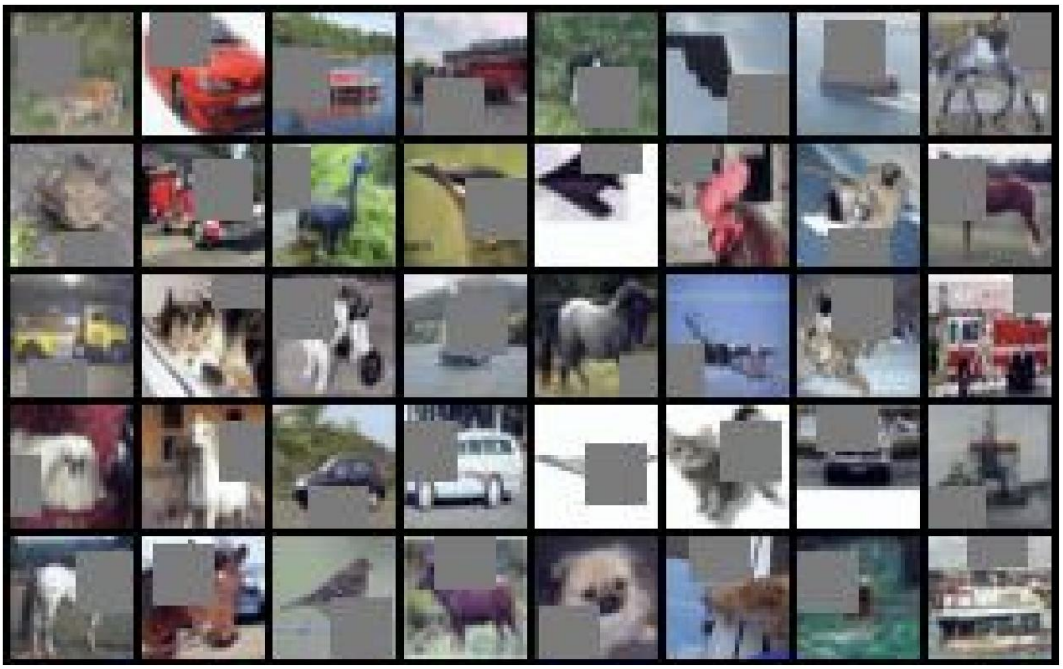


Figure 3.8 Example CutOut output [98]

- It makes our model do better detecting objects that are occluded (located) behind other objects.

- It encourages the model to learn more distinguishing features about each class of object. For example, if we are trying to detect American flags, the model may hone in on the stars. By covering up the stars in some of the images, you force it to also learn about the stripes. And thus, its performance in detecting American flags does much better when they are flapping in the wind.

D. CutMix data augmentation:

Combine images by cutting parts from one image and pasting them onto the augmented image. CutOut of the image force the model to learn to make predictions based on a robust number of features. In CutMix, the CutOut is replaced with a part of another image along with the second image's ground truth labelling. The ratio of each image is set in the image generation process (for example, 0.4/0.6). In Figure 3.9 below, we can see how the authors of CutMix demonstrate that this technique can work better than simple Cutout.

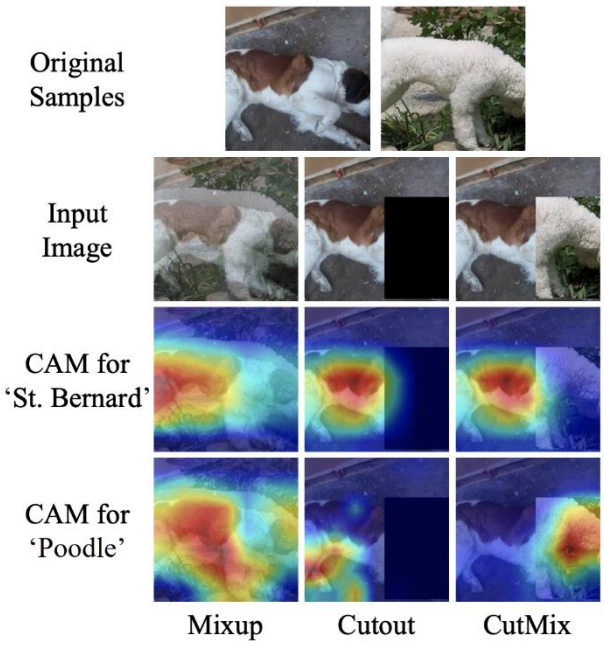


Figure 3.9 CutMix demonstration [66]

In short, with the release of YOLOv4 in 2020, we are using a better object detection network with new advanced data augmentation techniques**.**

3.1.2.2 Application phase

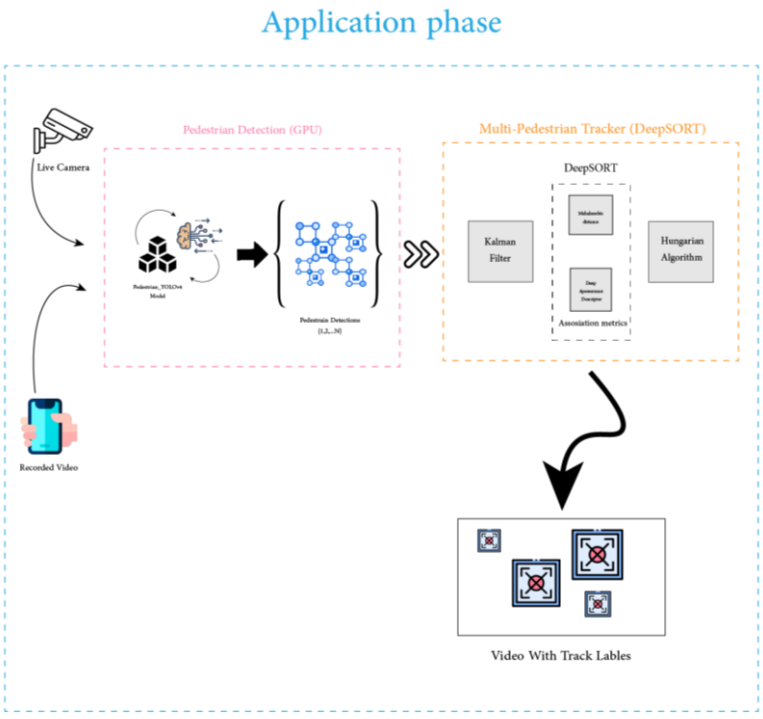


Figure 3.10 Application Phase

3.1.2.2.1 - Video Source Module

The Camera module provides the input (sequence images) to the rest of the system. These images can be obtained using four different sources:

- **Local camera (with OpenCV):** The Camera can read from a local camera using the OpenCV routine `VideoCapture ()` indicating the device number of the camera
- **Local video:** to read from a local video the OpenCV routine VideoCapture is also used but indicating the video path in the configuration file.
- **Local image files:** the path containing all the image files is required to be passed to VideoCapture. This video source is very useful because most of the datasets are provided as sequences of frames instead of videos. And it can avoid problems such as creating sequences of videos with the wrong duration or frame rate

3.1.2.2.2 – Pedestrian Detection on GPU

After Training and getting the new Parameters of the modified YOLOv4 to become a binary classifier (Pedestrian or not pedestrian) on Google Collab, we **re-implement** the Pedestrian_YOLOv4 with TensorFlow, to save the model as a TensorFlow model to be able to do the parallel Computing with during the application phase.

Thanks to the **TensorFlow 2.2** update that happened in January 2020, we can now save and load custom model defined by the users not just the ones available in the TensorFlow library, in order to be able to run "Pedestrian_YOLOv4" model in a GPU environment during the application phase, with the help of the Nvidia Toolkit called CUDA, we need to **re-implement** all YOLOv4 layers with the modification we need **(**Explained in ***3.1.2.1.3– Training Pedestrian_YOLOv4***), in order to make this model only focuses on detecting pedestrians to feed it to the DeepSORT algorithm in the fastest way possible.

Note: in this project we used the ***TensorFlow-GPU 2.3*** version during Application phase to do the calculation.

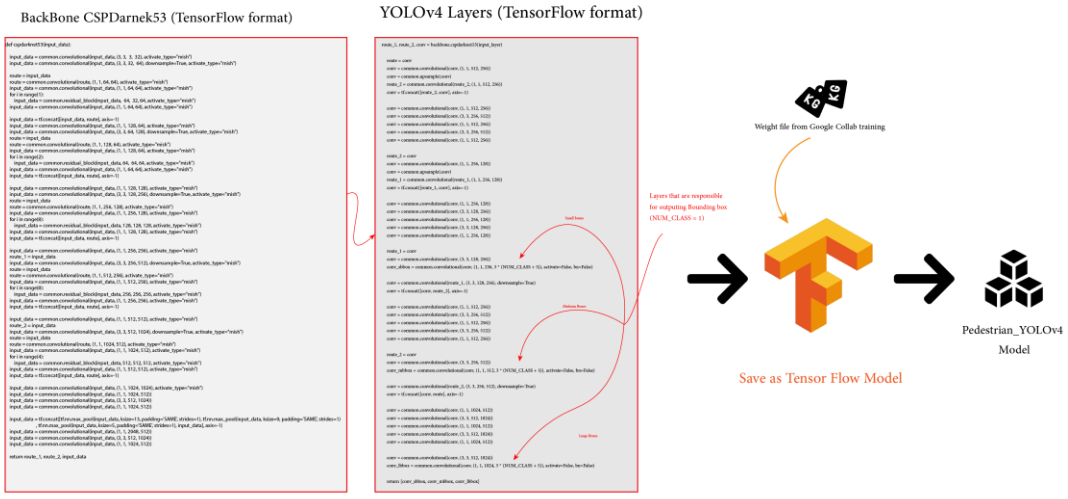


Figure 3.11 Pedestrian_YOLOv4 TensorFlow Model.

3.1.2.2.3– Multi-Pedestrian Tracking with DeepSORT

Now we have our pretrained model *Pedestrian_YOLOv4* and ready to detect objects and to be used by the tracking Algorithm, before we dive in to details of the algorithm's main components like the Kalman filtering [62] and the Hungarian algorithm [63]. we first see the general flow of how DeepSORT [59] works (Simple Online Realtime Tracking with Deep associating metric).

3.1.2.2.3.1- Core Process

Before DeepSORT was proposed, it was the SORT algorithm [62], but for the problem of identity switches (***IDSW**, see **chapter1, Object tracking metrics***), it only adopts the matching method of the distance between the tracker box and the detected box, without considering the what's in the box, so identity transformation is more likely to occur a lot, but this problem has been solved in the 2017 by DeepSORT [63].

First look at DeepSORT's Core process：

Prediction ⇒ (track) *Observation* (detection + data association) ⇒ *Update*

(See Figure 3.12).

Prediction: Predict the bounding box of the target in the next frame, (if we are using detections from series of **.txt** files, the Kalman filter tries to Predict the next (x, y) of the object which is not our case, because we are counting on our pre-trained model "Pedestrians_YOLOv4").

Observation (Detection): Perform target detection on the current frame with "Pedestrian_YOLOv4", the detections of object does not correspond to the target in the previous frame, so data association is required.

Update: Both prediction Bounding box and detection Bounding box will have errors, so the Kalman filter needs to update to minimize the errors in the future frames (*it will be explained later on this chapter*).

Schematic diagram of the core process:

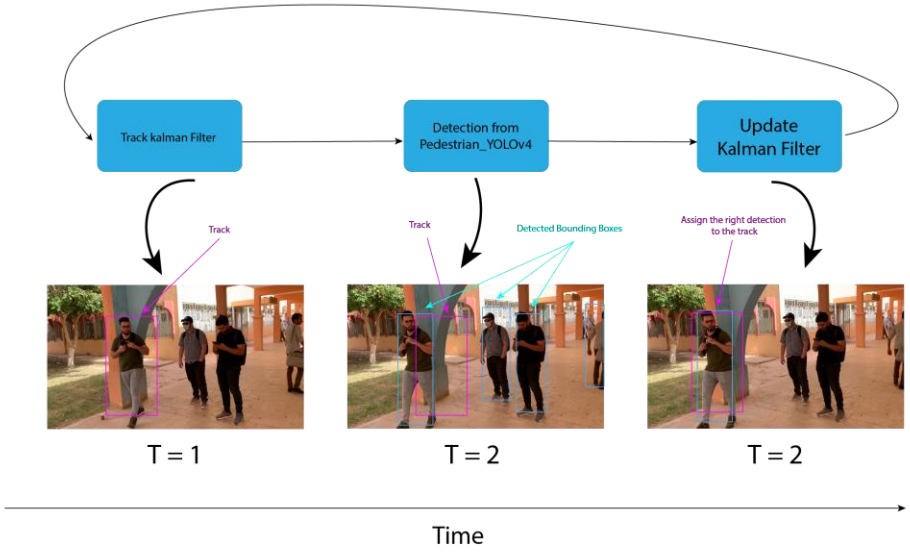


Figure 3.12 DeepSORT's semantic diagram of the core process.

As shown in the figure 3.12, after the track predicts the trajectory of a pedestrian at time **T=1** (purple box in Figure 3.12), the detection at **T=2** detects the objects in the picture and detects four targets (the cyan box in Figure 3.12). Through data association, the track Bounding Box (the purple box in from **T=1**) is tracked. Because it corresponds to the detection of **T=2.**

It is also at time **T=2**, after determining the association relationship, update the result of the track prediction, and use the Bounding box of detected object to represent the Bounding box of object tracking at **T**=2, Or an average Point between them (in our case we use the bounding box coordinate that the detector "YOLOv4" outputs).

3.1.2.2.3.2 – General Workflow

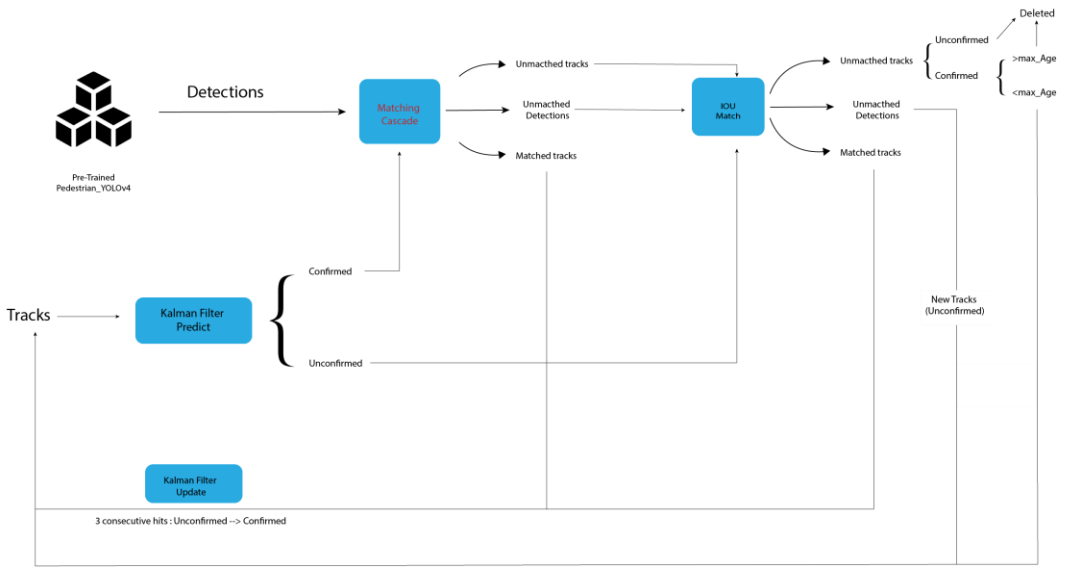


Figure 3.13 DeepSORT general workflow

➢ **Step I (Matched Tracks)**

We noted ***tracks*** at the beginning because at each frame set of tracks is selected to form a group of tracks instead of saying ***"track".*** (See Figure 3.14)

i) After Kalman filtering a trajectory of a given track's bounding box is predicted (predicting the location of a track in the current frame), two outputs, *confirmed* track and *unconfirmed* track (the uncertainty of an object we are tracking is the same or not).

ii) Perform the current frame detections using "Pedestrian_YOLOv4" and then associate the results of the predicted tracks to the detected boxes with *matching cascade* (explained later on this chapter).

iii) after the matching is completed update tracks.

Note: it should be noted that the update and matching are not delayed in time (for example: not like match at **T=2** and update in **T=3**, everything is done in one frame)

And then we loop through i), ii), iii)

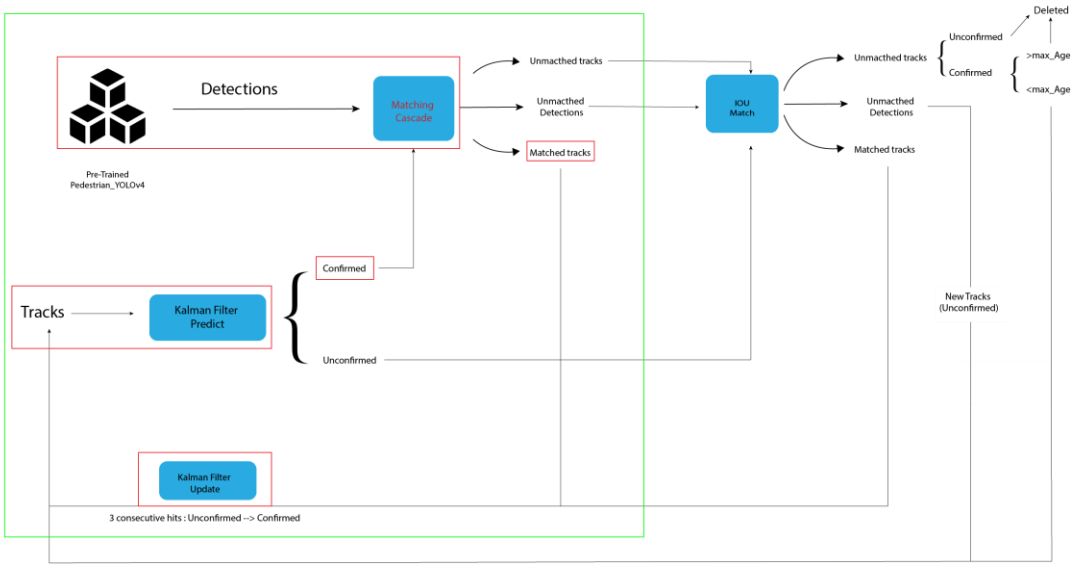


Figure 3.14 Matched Tracks.

➢ **Step II (Unmatched Tracks, Unmatched Detection)**

The *matching cascade outputs* three sets {*Matched Tracks, Unmatched Tracks, Unmatched Detections*}. (See Figure 3.15).

Why does the *Track* fail?

At certain frames the predicted track cannot be associated to a detection duo to the absent of a detection of the target.

Why does *Detection* match fail?

The existing tracks cannot be associated to a detection, let's say we have 3 tracks in frame **T=1** and a new object is detected at **T=2**, duo to the absence of this detection in the previous frames therefor it was not predicted by Kalman filtering.
As we saw the *matched Tracks* will go through an update, and the *Unmatched Tracks* and *Unmatched Detections* match again with IOU match (Intersection over Union using the Hungarian Algorithm (minimum cost matrix)).

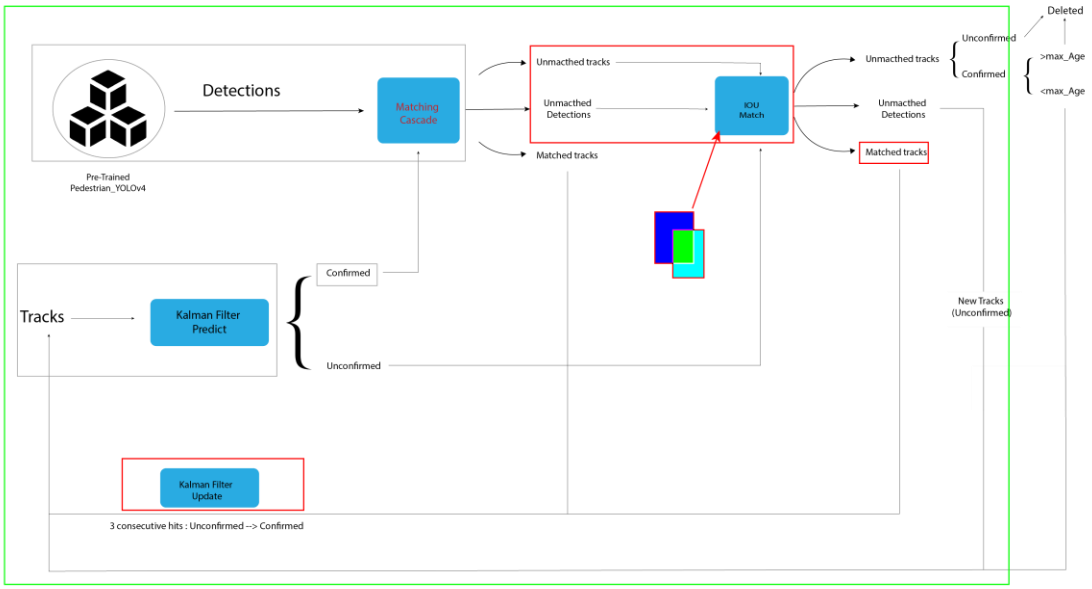


Figure 3.15 Unmatched Tracks, Unmatched Detection

- **Step III (New Tracks to the Unmatched Detection)**

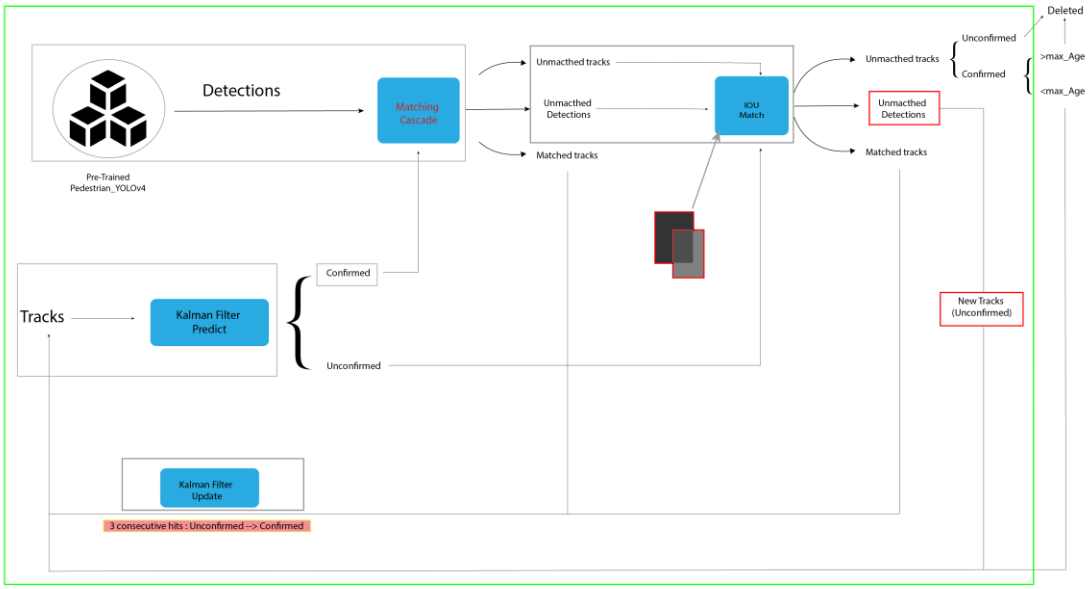


Figure 3.16 New Tracks to the Unmatched Detections.

The unmatched detections After the *IOU match*, they will be assigned a new track identity with *Unconfirmed Track* status, to skip the *matching cascade*, in case of a false detection alarm, and conduct three inspections if the track is outputted as a match track, change its status to confirmed to inter the *matching cascade*. (See Figure 3.16).

➢ **Step IV (Deletion of Tracks)**

For *Tracks* that still can't be matched to a detection, there's two situations (See Figure 3.17):

- If the *Track*'s status is *Confirmed*, we give the *Track* an extra time.
  - ➢ if the $Track_{age} < Age_{max}$ keep it in the set of Tracks.
  - ➢ if the $Track_{age} >= Age_{max}$ delete the Track's ID.

- If the Track's status is Unconfirmed delete the Track's ID immediately.

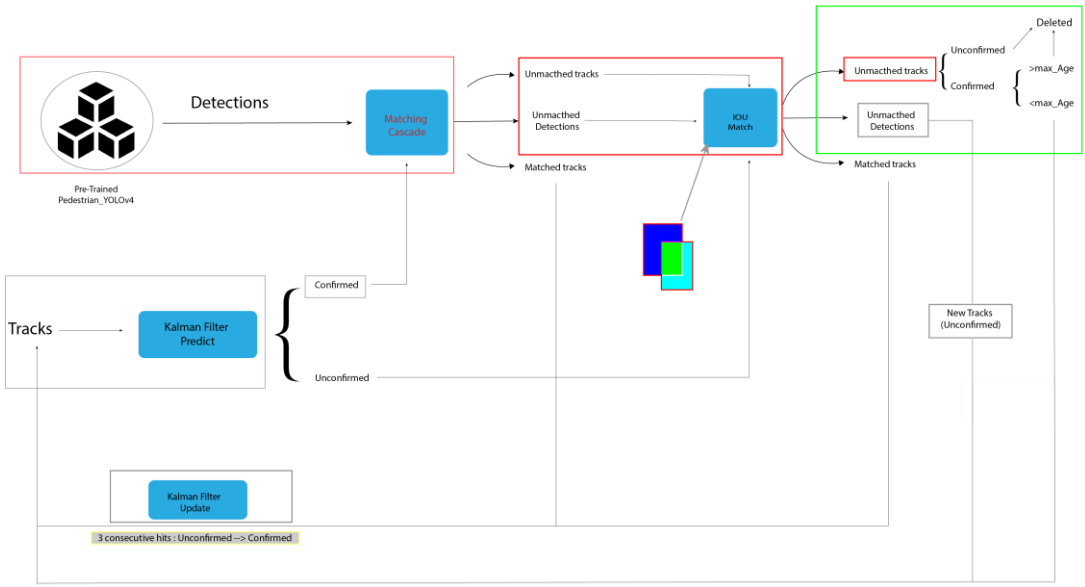


Figure 3.17 Deletion of Tracks.

3.1.2.2.3.3 - Detailed Workflow

A. Kalman Filter Framework

Kalman Filters [62], are very popular for tracking obstacles and predicting current and future positions. It is used in all sort of robots, drones, self-flying planes, self-driving cars, multi-sensor fusion, …etc.

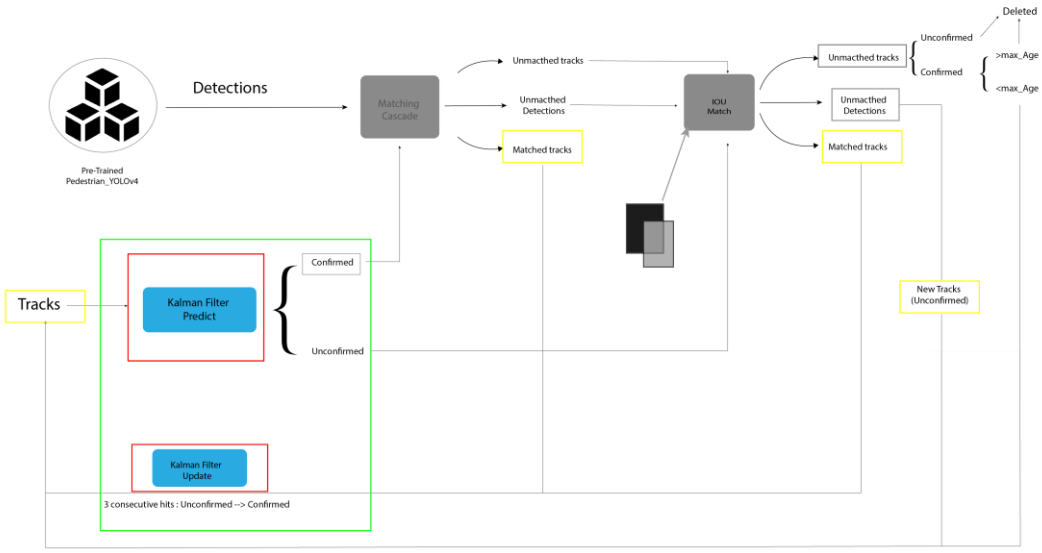


Figure 3.18 Kalman Filter Prediction and Update in Workflow

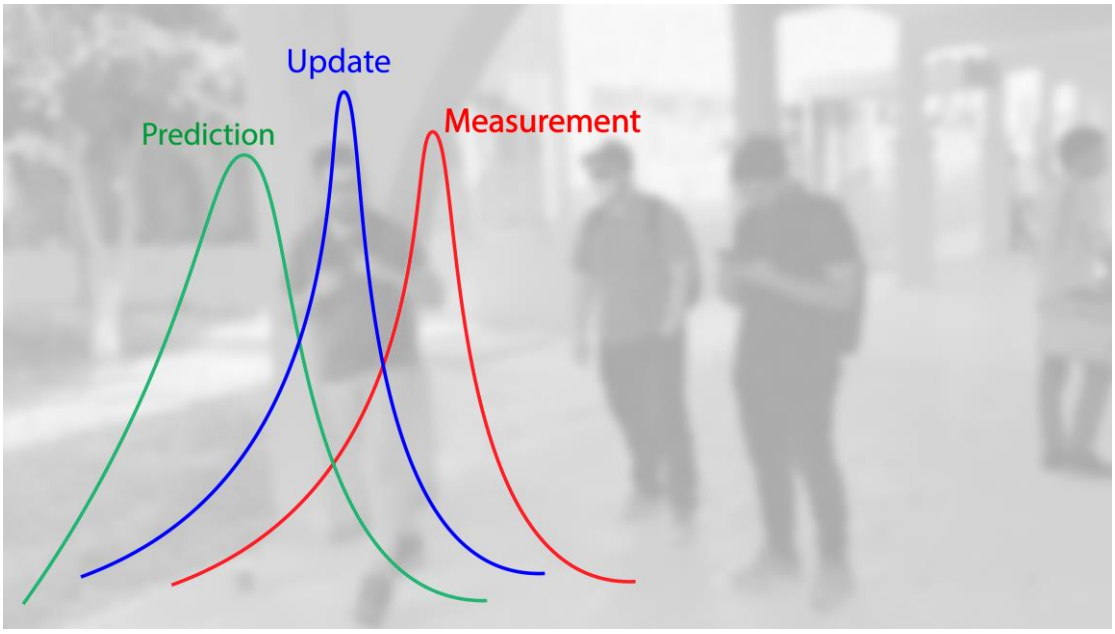


Figure 3.19 Kalman Filter Prediction and Update

A Kalman Filter is used on every bounding box, so it comes after a box has been matched. When the association is made, predict and update functions are called. These functions implement the math of Kalman Filters composed of formulas for determining state mean and covariance.

Note: Explaining the mathematical background of the next coming formulas will be beyond this master thesis scope.

- **State Mean and Covariance**

Mean and Covariance are what we want to estimate.

- Mean is the coordinates of the bounding box
- Covariance is our uncertainty on this bounding box having these coordinates.

- Mean (x) is a state vector. It is composed by coordinates of the center of the bounding box (cx, cy), size of the box (width, height) and the change of each of these parameters, velocities.

  $X = (Cx, Cy, w, h, vx, vy, vw, vh)$.
  When we initialize this parameter, we set velocities to 0 ($vx = vy = vw = vh = 0$). They will then be estimated by the Kalman Filter.

- Covariance (P) is our uncertainty matrix in the estimation. We will set it to an arbitrary number. A larger number means a larger uncertainty. (Example diagonal of 10)

$$P = \begin{bmatrix} \mathbf{10} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{10} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{10} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{10} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \mathbf{10} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \mathbf{10} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{10} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & \mathbf{10} \end{bmatrix} \quad (3.1) \text{ Covariance matrix}$$

Prediction will predict future positions; update will correct them and enhance the way we predict by changing uncertainty. With time, a Kalman Filter gets better and better to converge.

Example:

At time **T=0**, we have a measurement of 3 bounding boxes. The Hungarian Algorithm [63] (*will be explained later in this chapter*) defines them at 3 new detections. We therefore only have 3 detections in our system. For each box, we initialize Kalman Matrices with coordinates of the bounding boxes.

At time **T=1**, we have 3 bounding boxes, of the same objects. The Hungarian Algorithm matches them with the 3 former boxes and we can start calling predict and update. We predict the actual bounding boxes at time **T** from the bounding boxes at time **T-1** and then update our prediction with the measurement (with the location of the latest detection of a given Track) at time **T**.

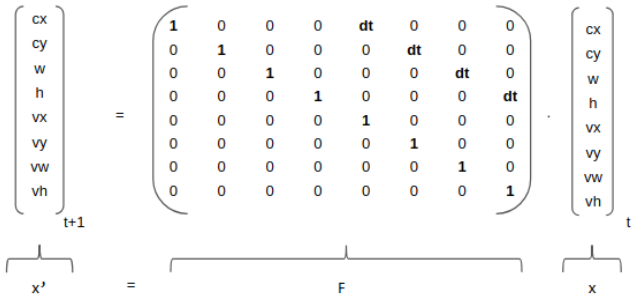
A.1-Kalman Prediction

Prediction phase is matrix multiplication that will tell us the position of our bounding box at time **T** based on its position at time **T-1**.

1. State Transition: F

F is the core implementation. What we put here is important because when we will multiply X by F, we will change our X and have a new X, called X', we estimate the X state at time **T**, based on the tracking state X at **T-1.** With the next formula:

$$x' = Fx \quad (3.2)$$



(3.3) Prediction x matrix Formula by Kalman Filter

F [8x8] matrix contains a time value: **dt** is the difference between current frame and former frame timestamp. We will have as result:

$cx' = cx + dt * vx;$
$cy' = cy + dt * vy;$
$cw' = cw + dt * vw \ldots etc$
$X' = (Cx', Cy', w', h', vx, vy, vw, vh).$

2. Covariance Matrix: Q

Q [8x8] is our noise matrix. It is how much confidence we give in the system. Its definition is very important and can change a lot of things. Q will be added to our covariance and will then define our global uncertainty. We can put very small values in the initialization with 0.01 (this translate how uncertain we are of these predictions in the first frames) and change it with time.

3. Covariance matrix: P'

Since now we have F, Q, F' matrices, we predict the covariance matrix P' at time **T,** With the next formula:

$$P' = FPF' + Q \quad (3.4)$$

A.2- Kalman Update

1. Measurement Vector: Z

Z is the measurement at time **T**. We don't input velocities here as it is not measured, simply measured values.

Where vector Z is: $Z = [cx\ cy\ w\ h\ ]$

2. Measurement Matrix: H

H [4x8] is our measurement matrix, it simply makes the math work between all or different matrices. We put the ones according to how we defined our state, and its dimension highly depends on how we define our state. Since our track state is [1x8] dimension (*Reminder of how the track state's shape*: $X = (Cx, Cy, w, h, vx, vy, vw, vh)$)

$$H = \begin{matrix} \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \mathbf{1} & 0 & 0 & 0 & 0 \end{matrix} \quad (3.5)$$

3. Measurement Noise: R

R is our measurement noise [4x4] matrix; it's the noise from the sensor. For a LiDAR or RADAR, it's usually given by the constructor. Here, we need to define a noise for "Pedestrian_YOLOv4" algorithm, in terms of pixels. It will be arbitrary, we can say that the noise in terms of the center is about 1 or 2 pixels while the noise in the width and height can be bigger, let's say 10 pixels.

$$R = \begin{matrix} \mathbf{1} & 0 & 0 & 0 \\ 0 & \mathbf{1} & 0 & 0 \\ 0 & 0 & \mathbf{10} & 0 \\ 0 & 0 & 0 & \mathbf{10} \end{matrix} \quad (3.6)$$

Based on the detection detected at time **T**, correct the state of the track associated with it to get a more accurate result, we calculate these Formulas:

- $y = Z - Hx'$ (3.7)

Where H maps the mean vector x' of the track to the detection space. Then (3.7) calculates the mean error between the *Detection* and the T*rack.*

- $S = HP'\,H^T + R$ (3.8)

S is the standard deviation of the *Track* (3.8). where H^T is the transpose of measurement matrix of H.

- $K = P'\,H^T S^{-1}$ (3.9)

K called the Kalman Gain (3.9), which is used to estimate the importance of the error.

- $x = x' + Ky$ (3.10)
- $P = (1 - KH)\,P'$ (3.11)

And then update our mean vector X and the new covariance matrix (3.10), (3.11) respectively.

Note: As we see in the Kalman filter framework [62] use in the estimation vector with the width and the height and their velocities as $X = (Cx, Cy, \boldsymbol{w}, \boldsymbol{h}, vx, vy, \boldsymbol{vw}, \boldsymbol{vh})$. but DeepSORT authors uses $X = (u, v, \boldsymbol{\gamma}, \boldsymbol{h}, \dot{u}, \dot{v}, \dot{\boldsymbol{\gamma}}, \dot{\boldsymbol{h}})$. Where $\boldsymbol{\gamma}, \boldsymbol{h}$ denotes the aspect ratio and height respectively.

Note2: We can directly use the "Pedestrian_YOLOv4" output detections format once it's successfully associated with a track (the first four numbers), since it's the same as the Kalman Filter estimation model framework. (***x, y, width, heigh****, confidence score, ...*).

B. The Hungarian Algorithm (Kuhn-Munkres)

The Hungarian algorithm [63], also known as Kuhn-Munkres algorithm, can associate an object from one frame to another, based on a **score** a group of **scores**, so basically, we are trying to associate our set of tracks with the right *Detection* among all *Detections* coming from "Pedestrian_YOLOv4" for each individual Track.

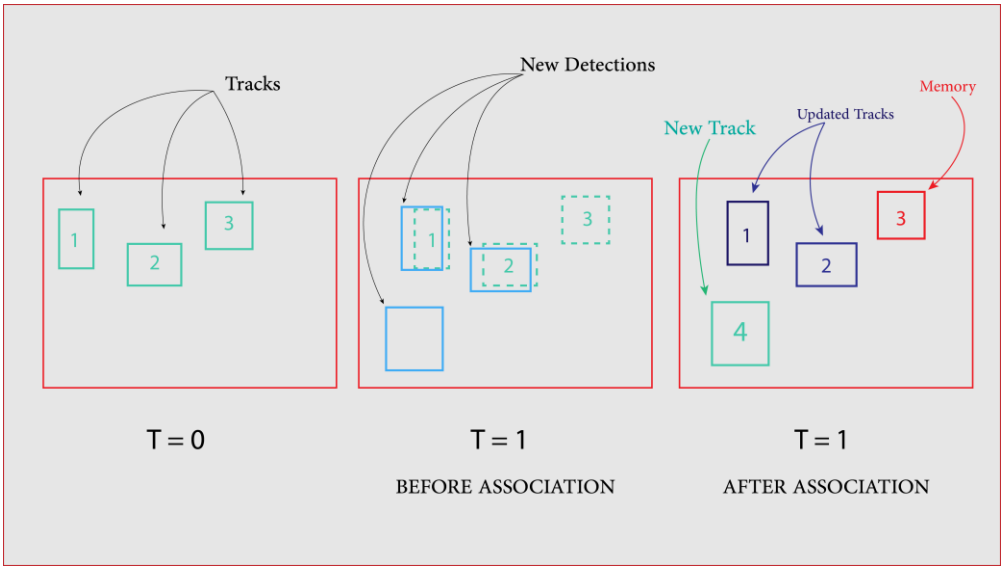


Figure 3.20 Data Association between Tracks and Detections.

We have many scores we can think of:

- **IOU (Intersection Over Union)**

  Meaning that if the bounding box is overlapping the previous one, so it's probably the same

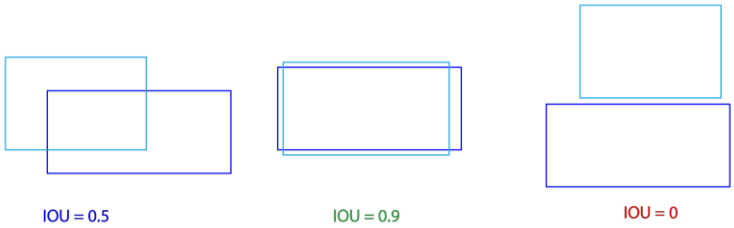


Figure 3.21 IOU scores

- **Convolution Cost (Cosine Distance)**

We could run a CNN (Convolutional Neural Network) like using the Re-identification model [59] to extract features from the bounding box and compare this result with the one from a frame ago. If the convolutional features are the same, then it means the objects looks the same. If there is a partial occlusion, the convolutional features will stay partly the same and association will remain. Like in DeepSORT uses this equation to calculate the similarity between two feature vectors from the Track and the Detection (*explained in* ***chapter 2, 2.2.2.2 – Appearance features***)

$$d^{(2)}(i,j) = \min\{1 - \boldsymbol{r}_j{}^{\mathrm{T}}\boldsymbol{r}_k^{(i)} \mid \boldsymbol{r}_k^{(i)} \in \mathcal{R}_i\}. \quad (3.1)$$

- **Mahalanobis Distance**

If we define the state of each Target's motion model as: $X = (u, v, \gamma, h, \dot{u}, \dot{v}, \dot{\gamma}, \dot{h})$.
(*Explained in* ***2.2.2.1 –Track managing & Estimation Model update***).
we have: (See Figure 3.22)

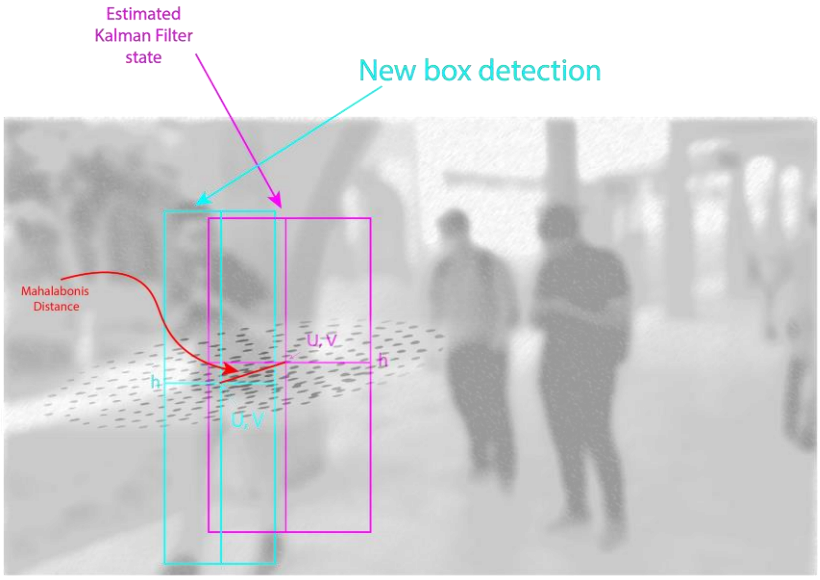


Figure 2.22 Mahalanobis distance (more the two points are in center, the more certainty of them to be the same object)

The Mahalanobis distance is a type of measurement like the Euclidian distance, but we can't rely on the Euclidean distance for three reasons:

i) U, V are is pixel-level.

ii) There is no magnitude between U, V of the track and U, V of the detected box by "Pedestrian_YOLOv4".

iii) We can't say two points are the same just depending on distance we have to consider also their motion state (Where the Object is going?).

The Mahalanobis distance takes into account the Covariance matrix S (Standard deviation of $Track_i$) by the Kalman Filter (3.8), to incorporate motion information, the covariance matrix indicates the uncertainty of the target information (the larger the number in the matrix the greater is the uncertainty that's why we inialize by diagonal of 10 in P, (See formula (3.1)).

Using the (squared) Mahalanobis distance between predicted Kalman states and newly arrived measurements with this formula (explained in details in ***2.2.2.2.2 – Motion Estimation***):

$$d^{(1)}(i,j) = (\boldsymbol{d}_j - \boldsymbol{y}_i)^{\mathrm{T}} \boldsymbol{S}_i^{-1} (\boldsymbol{d}_j - \boldsymbol{y}_i), \tag{2.2}$$

Where authors indicated that, if this distance is less than a certain threshold between the i-th track and j-th detection we can announce that the distance is admissible.

$$b_{i,j}^{(1)} = \mathbb{1}[d^{(1)}(i,j) \leq t^{(1)}] \tag{2.3}$$

Where $t^{(1)}$ = 9.4877

Based on these scores we can solve data association between tracks and detections optimally with linear assignment (Explained In, ***C. IOU Match***) by finding the minimum cost between each track and each new detection.

C. IOU Match

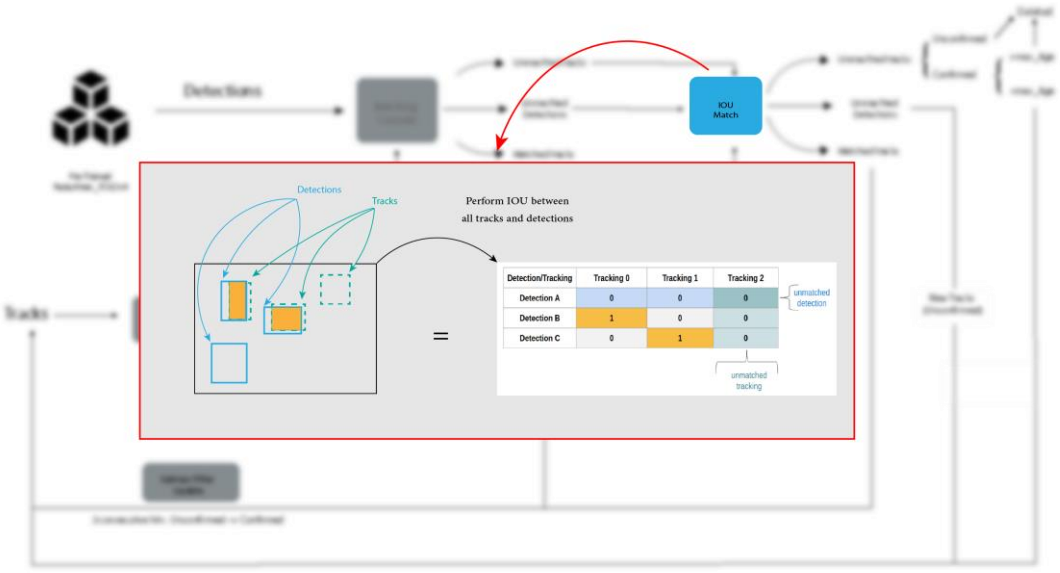


Figure 2.23 Inter-section-over-Union Match (IOU Match).

I. We have two lists of boxes:

1. A list of *Tracks* from **T=0**
2. A Detection list from **T=1**

II. We go through tracking and detection lists, and calculate IOU (*See the pseudo algorithm of Intersection-Over-Union, in chapter 1,* ***1.3.2.1.1 – Intersection over Union (IoU)***), and Store the IOU scores in a matrix …

III. Hungarian Algorithm steps: (assuming that the matrix is an NxN square matrix):

1. For each row of the matrix, subtract the smallest element.
2. For each column of the matrix, subtract the smallest element.
3. Cover all 0s in the matrix with the least horizontal or vertical lines.
4. If the number of lines is equal to N, the optimal allocation is found, and the algorithm ends, otherwise it goes to step 5.
5. Find the smallest element that is not covered by any line, subtract this element from each row that is not covered by the line, add this element to each column covered by the line, and return to step 3.

Example: In a giving time **T** with list of 3 detections, and list of tracks in time **T-1** and their IOU scores.

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	**0.15**	0.4	0.45
Detection 2	**0.2**	0.6	0.35
Detection 3	**0.2**	0.4	0.25

Step 1: The smallest elements in each row are ***0.15, 0.2, 0.2***, and subtracted to get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0.25	0.30
Detection 2	0	0.4	0.15
Detection 3	0	0.2	0.05

Step 2: The smallest element of each column is ***0, 0.2, 0.05***, and subtracted to get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0.05	0.25
Detection 2	0	0.2	0.10
Detection 3	0	0	0

Step 3: Cover all 0s with the minimum horizontal or vertical lines, and get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0.05	0.25
Detection 2	0	0.2	0.10
Detection 3	0	0	0

Step 4: The condition is not met (number of lines “2” less than N=”3”) goes to Step 5

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0.05	0.25
Detection 2	0	0.2	0.10
Detection 3	0	0	0

Step 5: Now the smallest element that is not covered is 0.05, and the rows that are not covered (the first and second rows) are subtracted by 0.05, and we get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	-0.05	0	0.20
Detection 2	-0.05	0.15	0.05
Detection 3	0	0	0

Add 0.05 to the covered column (the first column) to get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0	0.20
Detection 2	0	0.15	0.05
Detection 3	0.05	0	0

Jump to Step 3, cover all 0s with the minimum number of horizontal or vertical lines, and get:

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0	0.20
Detection 2	0	0.15	0.05
Detection 3	0.05	0	0

Step 4: The condition is met (number of lines =N= 3), and then we pull the maximum IOU scores

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0	0.20
Detection 2	0	0.15	0.05
Detection 3	0.05	0	0

- Track 1 for Detection 3
- Track 2 for Detection 2
- Track 3 for Detection 1

In some cases of overlapping bounding boxes, we can have two or more matches for one candidate. In this case, we set the maximum IOU value to 1 and all the others to zeros (in line or column)

Example: set the maximum IOU value to 1 and all the others to zeros

Detection/Tracking	Track 1	Track 2	Track 3
Detection 1	0	0	1
Detection 2	0	1	0
Detection 3	1	0	0

D. Cascade Matching

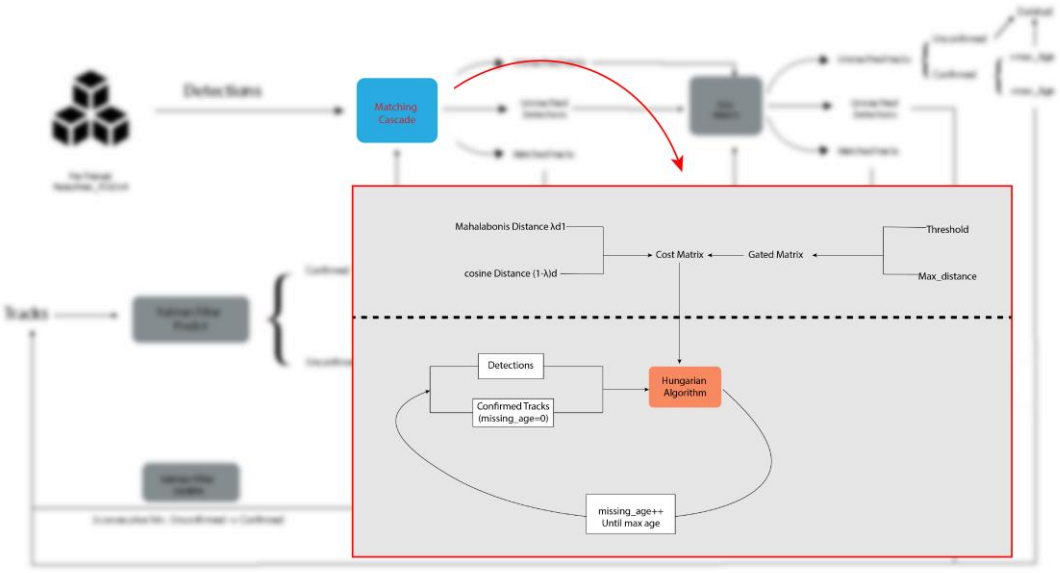


Figure 3.24 Cascade matching in the General workflow

The purpose of cascade matching is When a target is occluded for a long period of time, the uncertainty of Kalman filter prediction will be greatly increased, and the observability of the object will also be reduced. If two trackers compete for the matching the same detection, results at this time, the Mahalanobis distance of the trajectory with the longer occlusion time is often smaller, making the detection result more likely to be related to the trajectory with the longer occlusion time. This undesirable effect often destroys the continuity of tracking, assuming that the original covariance matrix is a normal distribution, then continuous tracks that are not updated will cause the variance of

this normal distribution to become larger and larger, then the point far away from the mean Euclidean distance may be separated from the previous distribution The closer the points get the same Mahalanobis distance value the bigger the probability of the match. Therefore, the author uses cascade matching to give priority to more frequently occurring targets. Of course, there are also drawbacks: it may cause some newly generated tracks to be connected to some old tracks. But this situation is rare.

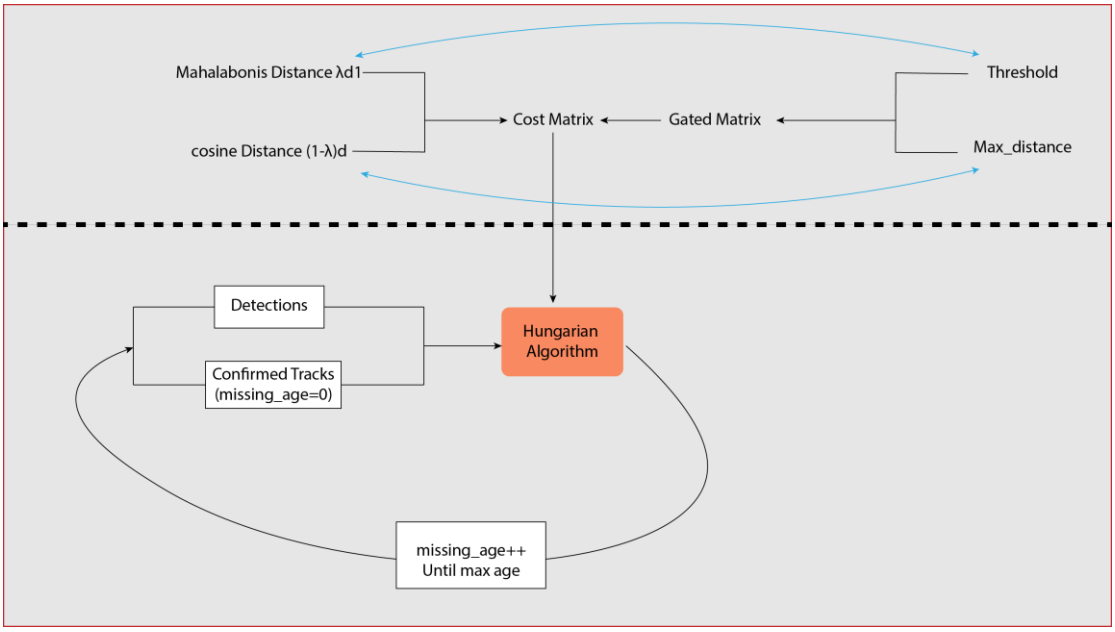


Figure 3.25 Cascade matching

- **Cost Matrix (Above the Dotted Line)**

Before we run the matching cascade first, we need to collect our scores

1. Mahalanobis distance between the prediction of the *Track*'s location with the Kalman filter and the *Detected* box (Discussed above in possible scores in the ***B. The Hungarian*** with:

$$d^{(1)}(i,j) = (\boldsymbol{d}_j - \boldsymbol{y}_i)^{\mathrm{T}} \boldsymbol{S}_i^{-1} (\boldsymbol{d}_j - \boldsymbol{y}_i), \tag{2.2}$$

2. The cosine distance between the *Track* and the *Detections* appearance features by taking out the last 128-dimension vector of the CNN's Re-Identification model trained offline (explained in details in ***2.2.2.2.5 - Deep Appearance Descriptor***).

Name	Patch Size/Stride	Output Size
Conv1	3 × 3/1	32 × 128 × 64
Conv2	3 × 3/1	32 × 128 × 64
Max Pool 3	3 × 3/2	32 × 64 × 32
Residual 4	3 × 3/1	32 × 64 × 32
Residual 5	3 × 3/1	32 × 64 × 32
Residual 6	3 × 3/2	64 × 32 × 16
Residual 7	3 × 3/1	64 × 32 × 16
Residual 8	3 × 3/2	128 × 16 × 8
Residual 9	3 × 3/1	128 × 16 × 8
Dense 10		128
Batch and ℓ_2 normalization		128

Table 3.2 DeepSORT Re-Identification network [67]

We calculate the Similarity (cosine) distance between the *Track* and the *Detection* with:

$$d^{(2)}(i,j) = \min\{1 - \boldsymbol{r}_j{}^{\mathrm{T}}\boldsymbol{r}_k^{(i)} \mid \boldsymbol{r}_k^{(i)} \in \mathcal{R}_i\}. \tag{2.4}$$

And then a weighted sum between the Mahalanobis and the cosine distance with:

$$c_{i,j} = \lambda\, d^{(1)}(i,j) + (1-\lambda) d^{(2)}(i,j) \tag{2.5}$$

Note: if the camera is not stable (moving) it is better to set λ=0, so we only count on the feature descriptor since we have no ego-motion information of the camera in the Mahalanobis distance.

- **Gated Matrix (Above the Dotted Line)**

the Mahalanobis distance has a bias towards predictions with larger uncertainty, since the predictions with larger standard deviations S (Formula 3.8), tend to have smaller $d^{(1)}$ (Equation 2.2) with the detection box. the assignments are valid only if they are within a gating region formed by two assignment metrics:

❖ For Mahalanobis distance: $b_{i,j}^1 = d^{(1)} \leq t^{(1)}$ (3.12), where $b_{i,j}^1$ returns a Boolean result (0 or 1)

❖ For Cosine distance: $b_{i,j}^2 = d^{(2)} \leq t^{(2)}$ (3.13), where $b_{i,j}^2$ returns a Boolean result (0 or 1)

❖ And finally, the Gate Matrix is: $b_{i,j} = \prod_{m=1}^{2} b_{i,j}^m = b_{i,j}^1 \times b_{i,j}^2$ (2.6), the assignments are valid if only they are Within the gating region (i.e. $b_{i,j} = 1$)

- **Cascade Matching (Below the Dotted Line)**

The cascade solves this problem by assigning bounding boxes to the younger age tracks first, where the age is defined as the number of frames since the first measurement (detection). When assigning the detections to tracks, we take the scores above (Cosine and Mahalanobis distances) to solve a minimum cost optimally via the Hungarian algorithm the same way of IOU score explained above (in ***2.2.2.3.2 - Detailed Workflow, C. IOU Match***).

Note: Matching cascade pseudo algorithm explained in ***2.2.2.2.4– Cascade Matching***

3.2– Project's Workflow Summary

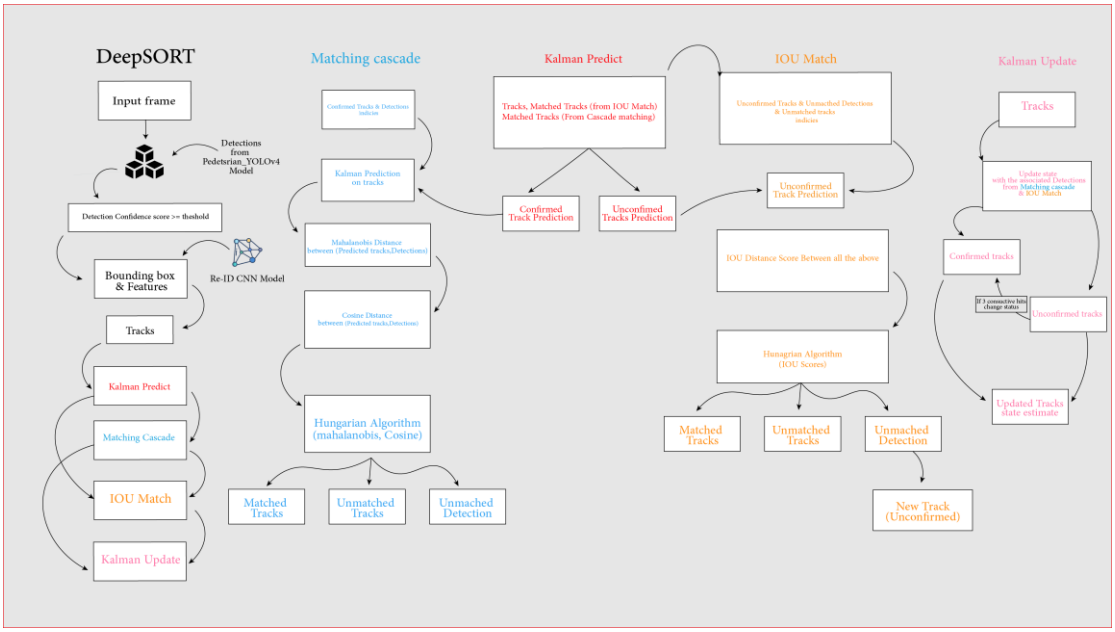


Figure 3.26 Workflow Summary

The Workflow in general starts with an input frame, we feed our "Pedestrian_YOLOv4" model with the frame to extract detections and then we run those output bounding boxes through another CNN model that extract features so we can use them later in data association process by the Hungarian algorithm, we initialize our set of tracks based on the detected objects then we predict these tracks' locations in within the frame based on the previous frame, then we go through ***Matching Cascade*** phase, and calculate scores (the Cosine CNN based distance, Mahalanobis distance Explained in ***2.2.2.3.2 - Detailed Workflow, B. The Hungarian Algorithm***) between N set of Tracks' predictions by Kalman filter and N set of newly arrived Detections by "Pedestrian_YOLOv4", then Based on these scores we can solve data association between Tracks and Detections optimally with linear assignment by finding the minimum cost between them (The Hungarian algorithm). The matched tracks from this process get ***a Kalman update*** by the correct detection for future accurate ***Kalman predictions*** of the motion of the track, then the unmatched tracks and the unmatched detection, get another chance by the ***IOU Match*** to see if there any leaks, which also uses the Hungarian algorithm but with the Intersection-Over-Union (IOU) score only, with also the tracks with unconfirmed state , the ***IOU Match***

outputs also {matched tracks, unmatched tracks and unmatched detections}, the matched tracks get a ***Kalman update***, if the unmatched track state is unconfirmed delete it immediately, if the unmatched track is confirmed give it a little time with A_{max} , till a correspondent detection occur, if not delete the track and consider it as lost, the unmatched detections get a new track with an unconfirmed state, if the track got associated with a detection 3 consecutive time change its state to confirmed.

Chapter 4: Implementation and Results

4.1- Implementation

4.1.1 – Environments and Developing tools

Since our system has multiple aspects, we are going to use various environments and tools including API's and libraries… etc, where are only going to site the major ones (***See 4.1.2-Environment details*** for more information).

- **Python**

Python is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991, Python's design philosophy emphasizes code readability with its notable use of significant whitespace. Its language constructs and object-oriented approach aims to help programmers write clear, logical code for small and large-scale, it was the main component to make this project, which incredibly made this project easy for us to implement.

Figure 4.1 Python Logo

- **TensorFlow**

TensorFlow is a free and open-source software library for dataflow and differentiable programming across a range of tasks. It is a symbolic math library, and is also used for machine learning applications such as neural networks. It is used for both research and production at Google.

TensorFlow was developed by the Google Brain team for internal Google use. It was released under the Apache License 2.0 on November 9, 2015.

To be more specific TensorFlow-GPU version.

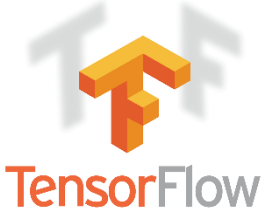


Figure 4.2 TensorFlow Logo

- **CUDA**

CUDA is a parallel computing platform and application programming interface model created by Nvidia. It allows software developers and software engineers to use a CUDA-enabled graphics processing unit for general purpose processing — an approach termed GPGPU.

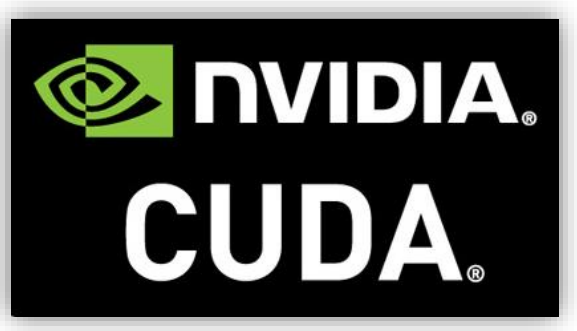


Figure 4.3 Nvidia CUDA Logo

- **OpenCV**

OpenCV is a library of programming functions mainly aimed at real-time computer vision. Originally developed by Intel, it was later supported by Willow Garage then Itseez. The library is cross-platform and free for use under the open-source BSD license.

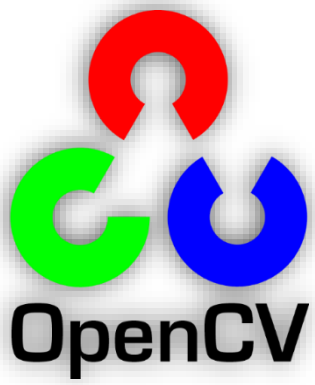


Figure 4.4 OpenCV Logo

- **PyCharm**

PyCharm is an integrated development environment used in computer programming, specifically for the Python language. It is developed by the Czech company JetBrains. Which incredibly help us to make the job with the libraries as easy as a click of a button.

Figure 4.5 Pycharm Logo

- **Anaconda**

Anaconda is a free and open-source distribution of the Python and R programming languages for scientific computing, that aims to simplify package management and deployment. Package versions are managed by the package management system.

Figure 4.6 Pycharm Logo

- **NumPy**

NumPy is a library for the Python programming language, adding support for large, multidimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate on these arrays.

Figure 4.7 NumPy Logo

- **Google Colaboratory**

Colaboratory, or "Colab" for short, allows us to write and execute Python scripts directly in the browser (Google Chrome, Opera, Firefox…etc), with zero configuration required, and Free access to a Free GPU. As of October 13, 2018, Google Colab provides a single 12GB NVIDIA Tesla K80 GPU that can be used up to 12 hours continuously. Recently, (Colab also started offering free TPU.) which saves us too much time for training the model.

Figure 4.8 NumPy Logo

4.1.2-Environment details

Package	Version	Latest Version
absl-py	0.12.0	0.13.0
astunparse	1.6.3	1.6.3
blas	1.0	1.0
ca-certificates	2021.5.25	2021.5.25
cachetools	4.2.1	4.2.2
certifi	2021.5.30	2021.5.30
chardet	4.0.0	4.0.0
cudatoolkit	10.1.243	11.0.221
cudnn	7.6.5	7.6.5
cycler	0.10.0	0.10.0
easydict	1.9	
freetype	2.10.4	2.10.4
gast	0.3.3	0.4.0
google-auth	1.30.0	1.32.0
google-auth-oauthlib	0.4.4	0.4.4
google-pasta	0.2.0	0.2.0
grpcio	1.37.0	1.36.1
h5py	2.10.0	3.2.1
hdf5	1.8.20	1.12.0
icc_rt	2019.0.0	2019.0.0
icu	58.2	68.1

idna	2.10	3.2
importlib-metadata	4.0.1	3.10.0
intel-openmp	2021.2.0	2021.2.0
jpeg	9b	9b
keras-preprocessing	1.1.2	1.1.2
kiwisolver	1.3.1	1.3.1
libopencv	3.4.2	4.0.1
libpng	1.6.37	1.6.37
libtiff	4.1.0	4.2.0
lxml	4.6.3	4.6.3
lz4-c	1.9.3	1.9.3
markdown	3.3.4	3.3.4
matplotlib	3.3.4	3.3.4
matplotlib-base	3.3.4	3.3.4
mkl	2021.2.0	2021.2.0
mkl-service	2.3.0	2.3.0
mkl_fft	1.3.0	1.3.0
mkl_random	1.2.1	1.2.1
numpy	1.18.5	1.20.2
numpy-base	1.20.1	1.20.2
oauthlib	3.1.0	3.1.1
olefile	0.46	0.46
opencv	3.4.2	4.0.1
opencv-python	4.1.1.26	
openssl	1.1.1k	1.1.1k
opt-einsum	3.3.0	
pandas	1.2.4	1.2.5
pillow	8.2.0	8.2.0
pip	21.0.1	21.1.3
protobuf	3.15.8	3.14.0
py-opencv	3.4.2	4.0.1
pyasn1	0.4.8	0.4.8
pyasn1-modules	0.2.8	0.2.8
pyparsing	2.4.7	2.4.7
pyqt	5.9.2	5.9.2
python	3.7.0	3.9.5
python-dateutil	2.8.1	2.8.1
pytz	2021.1	2021.1
qt	5.9.7	5.9.7
requests	2.25.1	2.25.1
requests-oauthlib	1.3.0	1.3.0
rsa	4.7.2	4.7.2

scipy	1.4.1	1.6.2
seaborn	0.11.1	0.11.1
setuptools	52.0.0	52.0.0
sip	4.19.8	4.19.25
six	1.15.0	1.16.0
sqlite	3.35.4	3.36.0
tensorboard	2.2.2	2.5.0
tensorboard-plugin-wit	1.8.0	1.6.0
tensorflow-gpu	2.3.0rc0	2.3.0
termcolor	1.1.0	1.1.0
tf-estimator-nightly	2.3.0.dev2020062301	
tk	8.6.10	8.6.10
tornado	6.1	6.1
tqdm	4.60.0	4.61.1
typing-extensions	3.7.4.3	3.10.0.0
urllib3	1.26.4	1.26.6
vc	14.2	14.2
vs2015_runtime	14.27.29016	14.27.29016
werkzeug	1.0.1	1.0.1
wheel	0.36.2	0.36.2
wincertstore	0.2	0.2
wrapt	1.12.1	1.12.1
xz	5.2.5	5.2.5
zipp	3.4.1	3.4.1
zlib	1.2.11	1.2.11
zstd	1.4.9	1.4.9

4.2 – Results

All results shown are calculated in an Online manner

4.2.1 – Experiment Setup

- **Local machine (Laptop):**

  - **GPU:** Nvidia MX150, 4G RAM
  - **CPU:** Intel i7 8th generation @1.80 GHz
  - **RAM:** 8G

- **Google Colab Virtual Machine:**

  - **GPU:** Nvidia Tesla K80, 12GB / 16GB
  - **CPU:** Xeon Processors @ 2.3 GHz
  - **RAM:** 12.6 GB

4.2.2 – FPS Results

Since our project is built to work in two environments, **CPU** based and **GPU** based, depending on the TensorFlow installed in the environment (*TensorFlow-CPU* or *TensorFlow-GPU*).

- **Google Colab:**

Tested on the same video source, because the performance depends on how many objects being tracked in the scene.

	GPU	**CPU**
Yolov4_DeepSORT	20.24 ~ 27.07 FPS	0.6 ~ 0.62 FPS
Pedestrian_YOLOv4(ours)	**20.27 ~ 28.8** FPS	0.65 ~ 0.67 FPS

Table 4.1 FPS comparison with Google Colab

- **Local Machine (My Laptop)**

	GPU	CPU
Yolov4+DeepSORT	1.8 ~ 4.34 FPS	0.6 ~ 0.75 FPS
Pedestrian_YOLOv4+DeepSORT(ours)	**1.9 ~ 4.98** FPS	**0.7 ~ 0.9** FPS

Table 4.2 FPS comparison with Local Machine

As we can see we hit the score of 28.8 FPS on Google Colab is Realtime. (The FPS results ca be improved by stronger GPU like Nvidia RTX2080 Ti)

4.2.3 – Results at Different Scenarios

4.2.3.1 – Walking on street (Camera moving)

One of the famous Situations when walking on streets are when a person occludes another and the occluded target needs to get re-identified, where the motion prediction gets a little bit tricky, but our work handles that quite well, in Figure 4.9 shows frames of in hakim Saadane Biskra, where I was walking and filming. (Challenging situation for algorithms that has no camera ego motion consideration like DeepSORT)

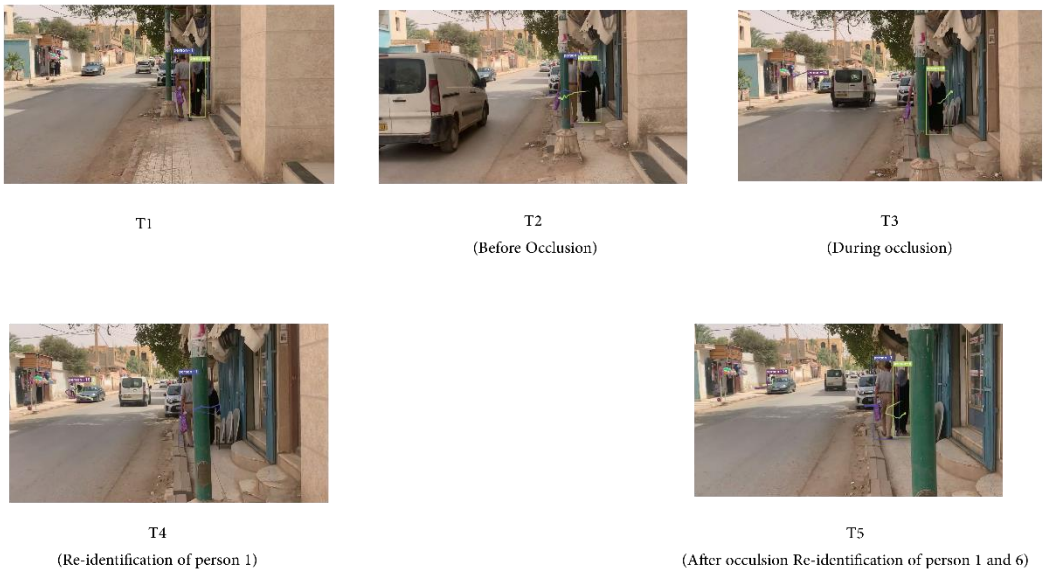


Figure 4.9 Video captured in Hakim, Saadane, Biskra (With our method).

4.2.3.2 – Football live Match

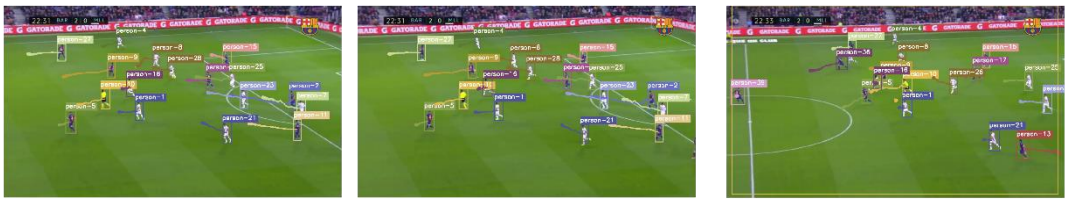


Figure 4.10 Football match tracking (with Our method).

Duo to the algorithm ability to track in an online manner where it only uses the current and previous frames for motion prediction, object tracking and the real-time FPS calculations, it is applicable for live football matches scenarios (Figure 4.10).

4.2.3.3– Live Webcam

- **With Good Lighting**

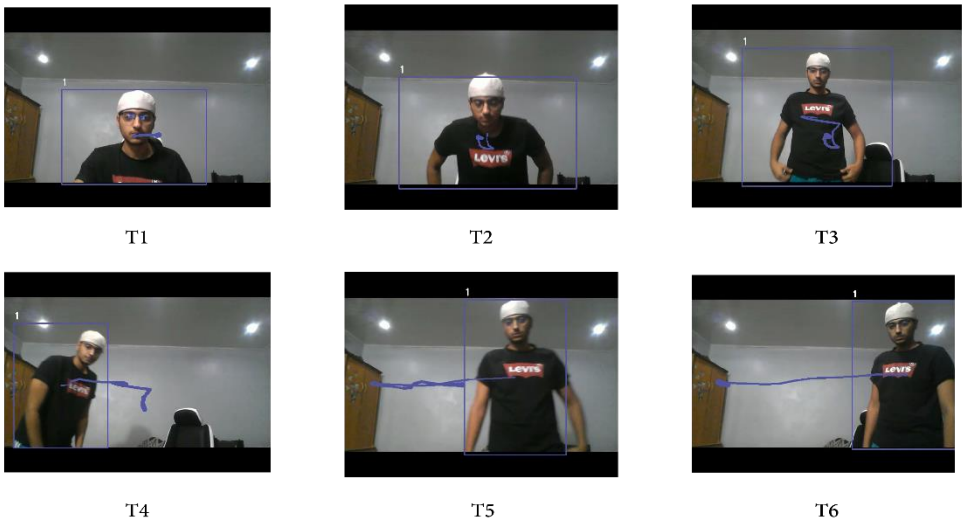


Figure 4.11 Live Webcam Results with good lighting (with our method)

In Figure 4.11 the tracker works also good on live scenarios, like in webcam, which is really challenging duo to medium quality we feed our tracker, regardless it successfully detected and tracked the person (me) successfully while traveling left and right in the room.

- **With Low Lighting condition**

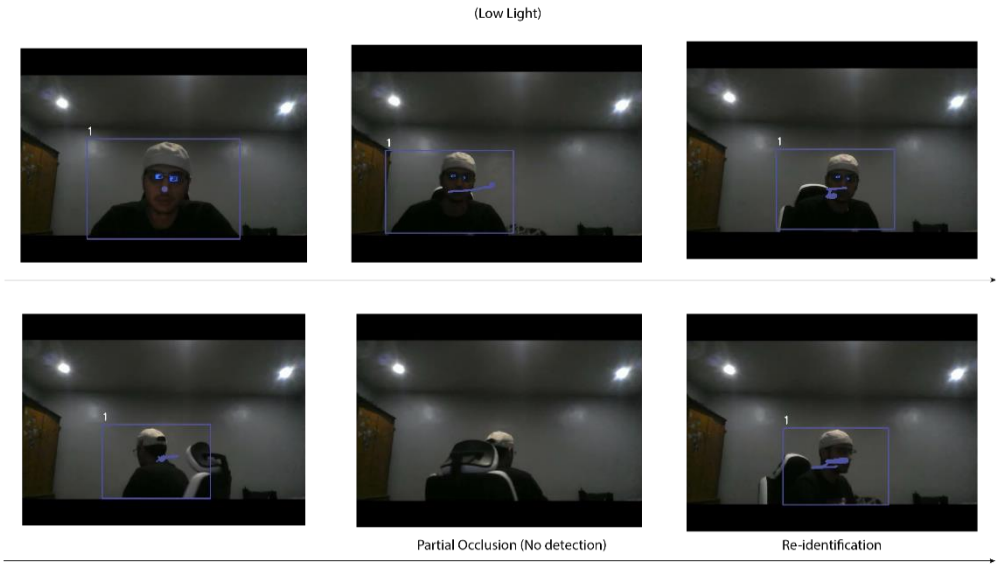


Figure 4.12 Live Webcam Results with low lighting (with our method)

As seen in Figure 4.12 we even apply more challenging conditions with turning off the bright lights and keeping the ones behind to see if the tracker can come through these conditions, and we can observe our tracker it manages once again to successfully detect and track the Person in the camera.

even though I was partly occluded by the chair, our work managed to re-identify what's Infront him duo to the Re-ID CNN model that do the feature extraction of the detected objects.

4.2.3.4 – Fixed Camera

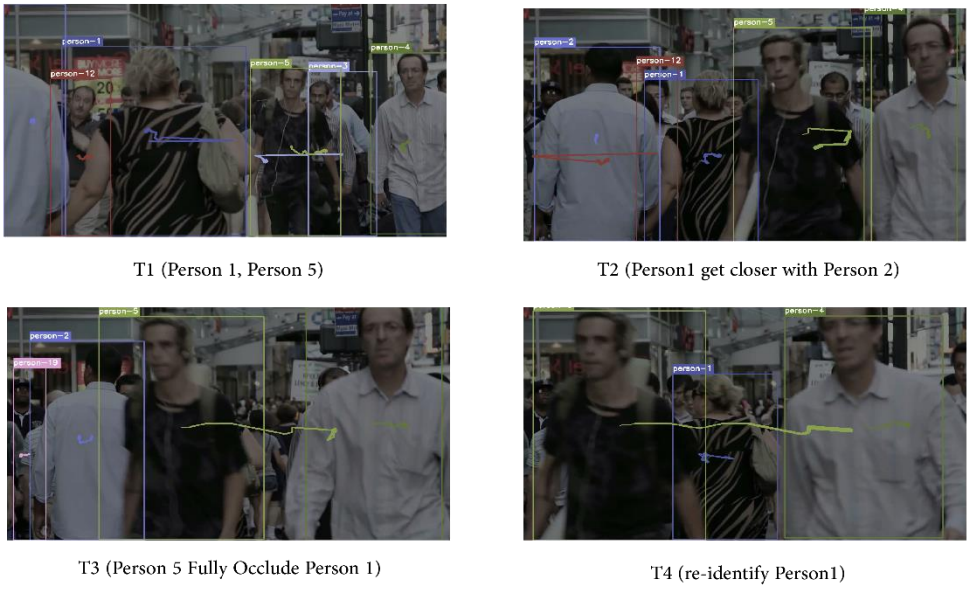


Figure 4.13 fixed camera on street with (Our method)

We even did experiments with YouTube videos, where the situation is a scene with fixed camera and closer angle to pedestrians (occlusion occur more in these types of situations (See Figure 4.13), our method also gives good results, and re-identify the pedestrians successfully.

Figure 4.14 CCTV security camera indoors placement (with our method).

The situation in Figure 4.14 is to place the camera angle in a corner or a ceiling (Security camera), where it captures all room (or a street), we can denote that the detection and the tracking gave also good results.

4.2.4 – Comparison in Night Time and Grayscale Cameras

As we know tracking pedestrians has many utilities, the first thing comes to mind is to apply it for security measurements purposes, and tracking pedestrian at night is as important as tracking during the day. Not all security cameras that are on the market are high quality resolution, and they come in

different shapes in terms of quality and colour output, there's also companies or home owners who has Grayscale security cameras, so we need a robust and reliable tracker that works in both situation (night & time), and also flexible for all cameras' quality and colours, the results were nearly the same during the day, but the YOLOv4 original pretrained model finds difficulties in detecting Pedestrians in a crowded Grayscale security cameras specially during the night.

Our "Pedestrian_YOLOv4" performs a lot better with DeepSORT during night time and low lighting conditions, duo to the custom dataset we collected and modifying the architecture to only focuses on pedestrian rather than all object classes in the scene, the model gives better results and better adaptability to different scenarios specially with fixed camera and low light conditions.

4.2.4.1 – Grayscale Security Camera.

In the next comparison we are going to turn off the ID name (Person tag) and keep only the bounding boxes, and differentiate the tracks by colours of the bounding boxes, so the results be clearer to see.

A. YOLOv4 with DeepSORT

As shown in Figure 4.15 a lot of tracks are missed in the scene (Red circles in Figure 4.15) duo to the bad detections of pedestrian, and that comes from YOLOv4 is a multi-object detector with multiple classes where the architecture wants to capture as much objects as possible in scene, which gives poor results in terms of pedestrian detection in grayscale. And we can see it also outputs a lot of identity switch duo to the bad performance of the detector in detecting the pedestrian class.

In this video example we turned off all YOLOv4 classes, and we just allowed the "Person" detection to come through, to feed the DeepSORT algorithm. (For comparison purposes).

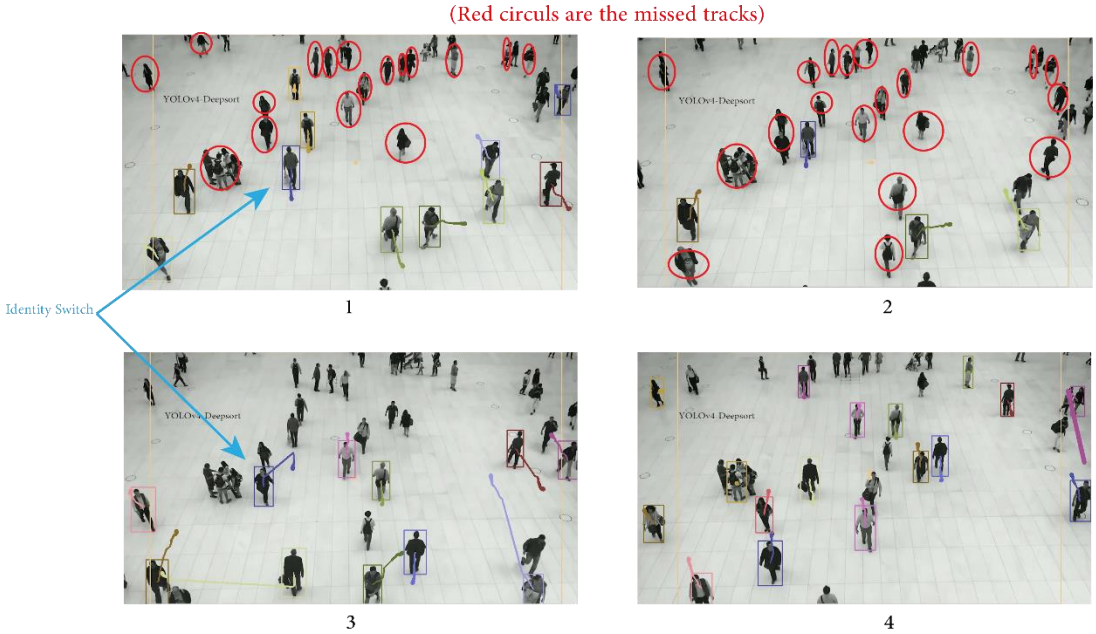


Figure 4.15 grayscale Airport Camera with "YOLOv4_DeepSOR

B. Pedestrian_YOLOv4 (Ours) with DeepSORT

Figure 4.16 Grayscale Airport Camera with “Pedestrian_YOLOv4_DeepSORT” (Ours)

As shown in Figure 4.16 (Zoom in to see clearly) the number of tracks, when now the architecture has been modified to just classify pedestrians in a given scene, and duo to the extra training on a custom dataset that we prepared the result are notably improved in such scenarios.

4.2.4.2- Extremely Low Light Conditions (MOT challenge Video)

In this experiment we noticed that even though the video from the MOT challenge dataset was captured at night, we see that the video is still high in terms of exposure and luminosity and that does not translate a real night condition, and duo to the lack of night security camera footage, we pre-processed the input video to simulate real night conditions

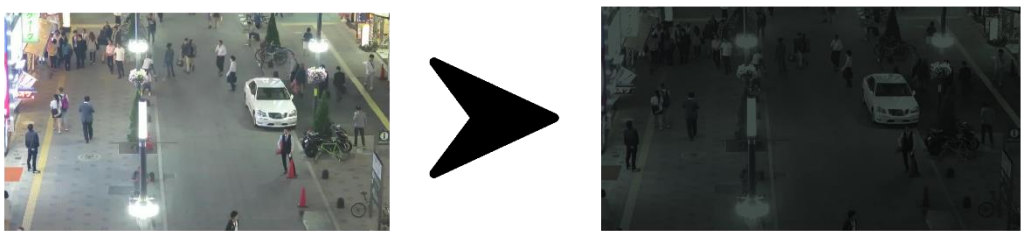


Figure 4.16 MOT challenge video pre-process to simulate night scene.

A. YOLOv4 with DeepSORT

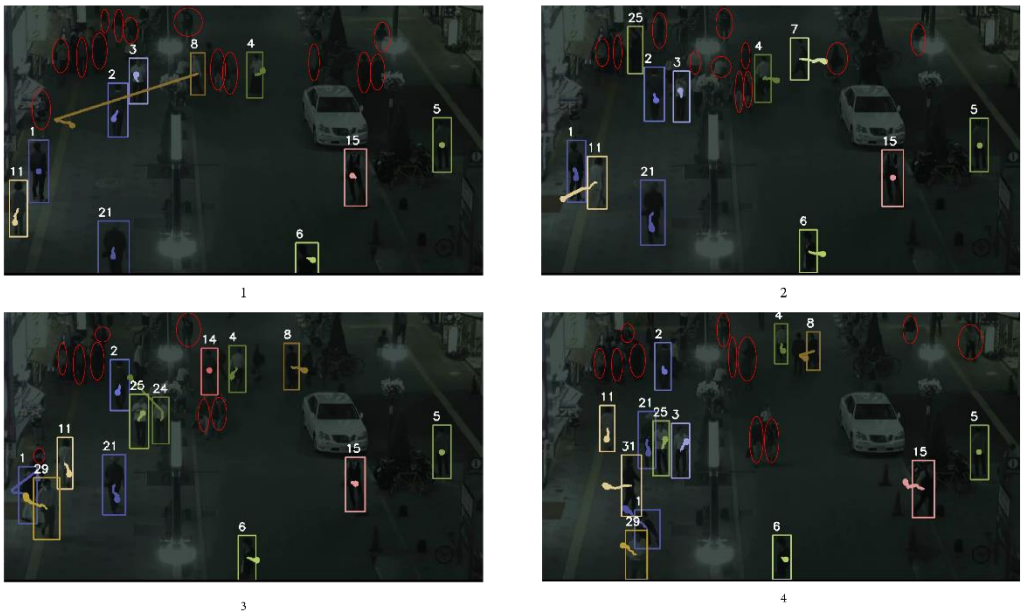


Figure 4.17 DeepSORT with the original YOLOv4.

As shown in Figure 4.17 (Zoom in to see clearly), there was a lot of missed detections (red circles) therefore DeepSORT couldn't create tracks and start tracking, that's duo to YOLOv4 instead of focusing in detecting Pedestrians.

B. Pedestrian_YOLOv4 with DeepSORT

As shown in Figure 4.18 (Zoom in to see clearly), now with our proposed work there's few pedestrians that are not being tracked (red circles) nearly all the detections were captured, and therefore DeepSORT performed better in terms of tracking, and that's duo to "Pedestrian_YOLOv4" architecture that instead of focusing in detecting all object in the scene, the architecture now focuses only on finding pedestrians in the scene. Which gives better results in terms of accuracy with keeping the calculations as fast as possible.

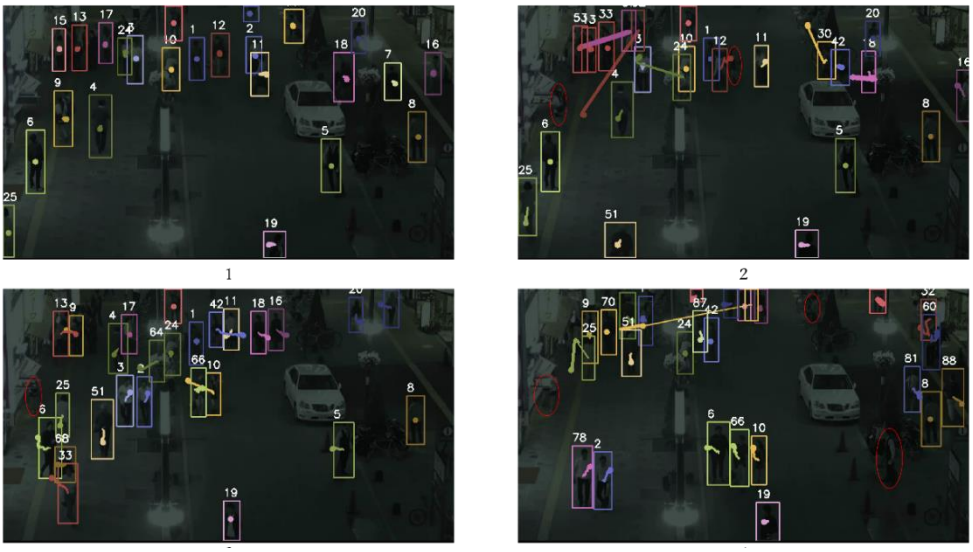
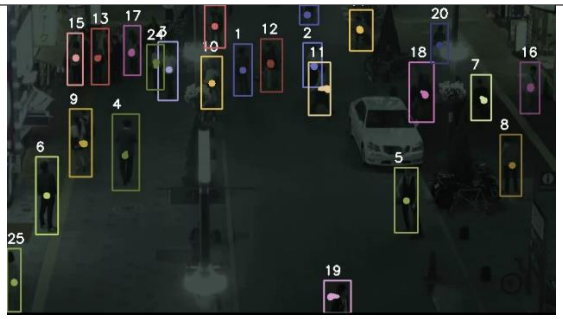
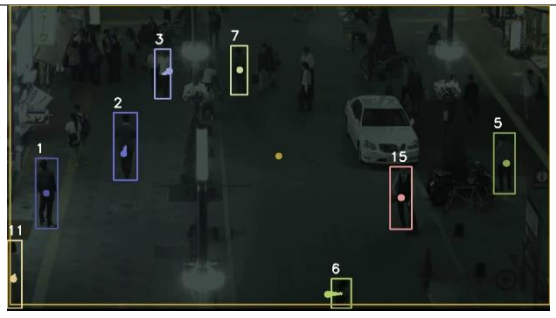
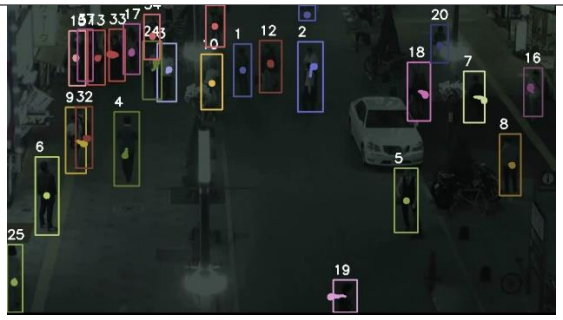
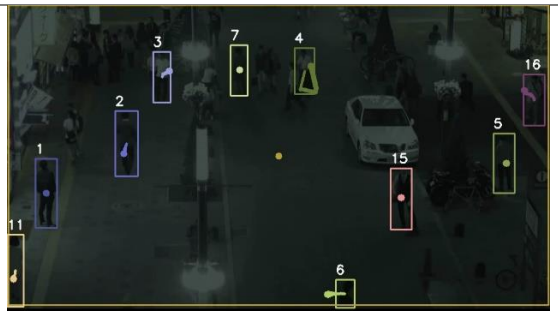
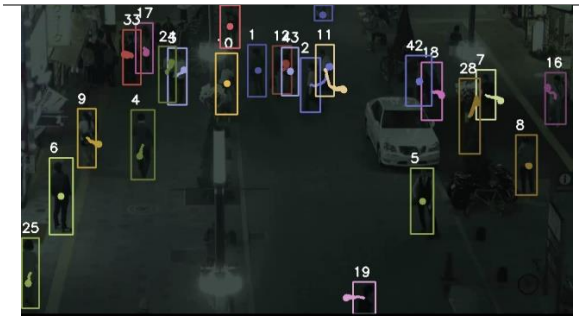
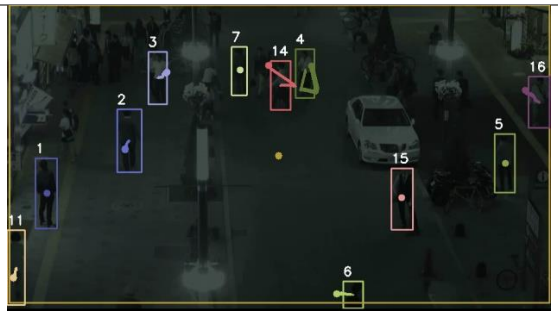
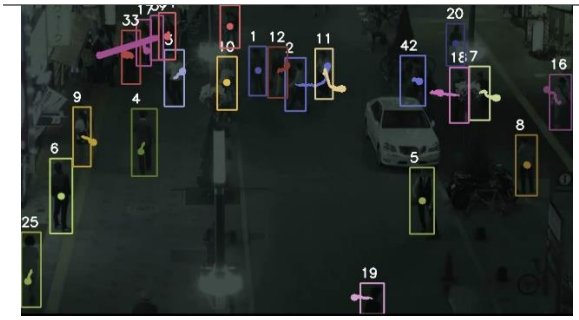
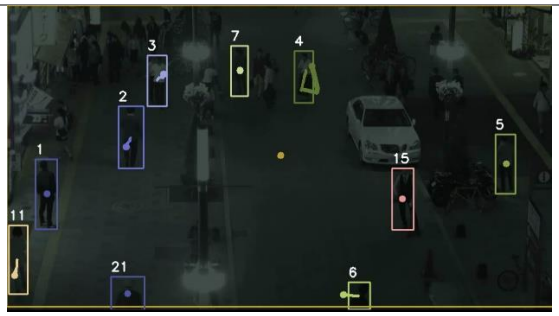
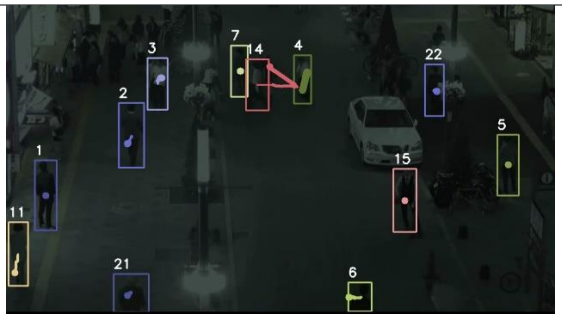
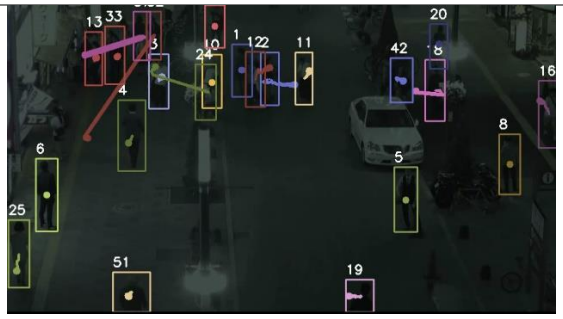
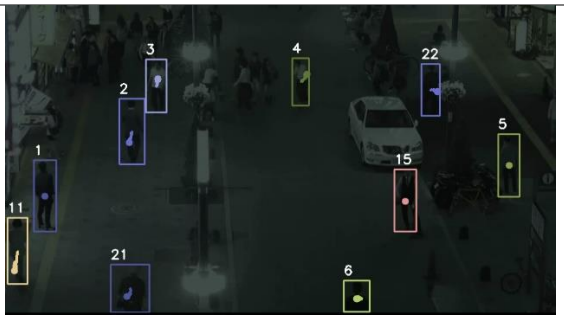


Figure 4.18 DeepSORT with the Pedestrian_YOLOv4 (Ours)

C. Detection Count Frame by Frame

In purpose to see results under scope and see the actual difference, we take an example of 3 second segment and take 10 frames as a time step, and compare detected pedestrians in both methods.

Pedestrian_YOLOv4_DeepSORT (Ours)	YOLOv4_DeepSORT	Frame
		1
		10

		20
		30
		40
		50
		60

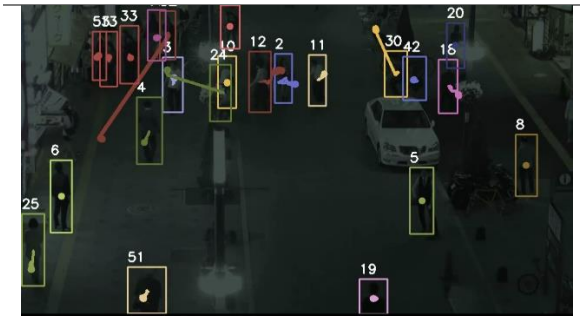
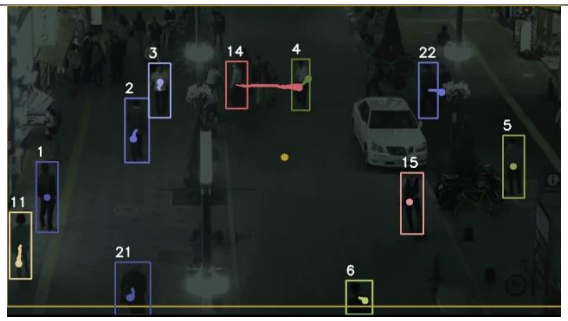
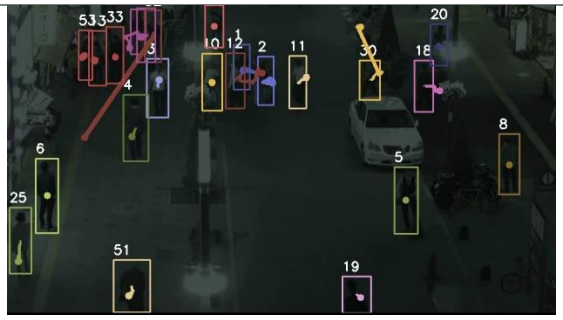
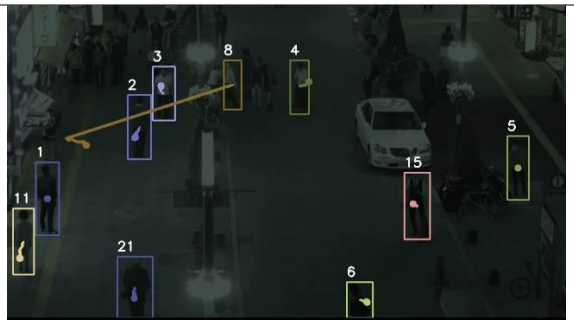
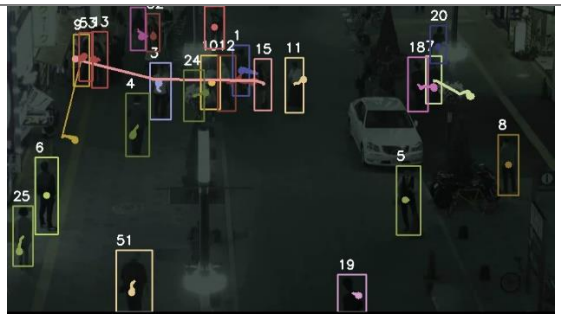
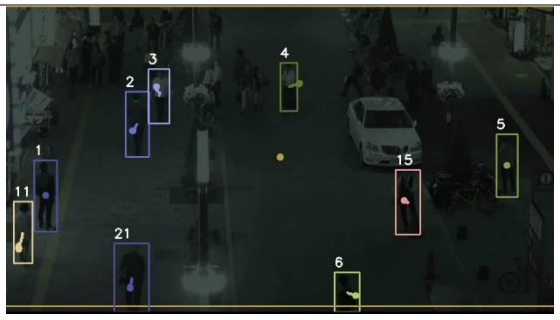
		70
		80
		90

Table 4.4 frame by frame Comparison

Frame	Detected	Not Detected	Total	Percentage	FPS (Tesla K80)
10	24	3	24/27	**88.88%**	**28.6**
20	25	1	25/26	**96,15%**	**28.8**
30	26	1	26/27	**96,23%**	**26.04**
40	24	6	24/30	**80%**	**26.8**
50	23	2	23/25	**92%**	**26.8**
60	25	2	25/27	**92.59%**	**26.04**
70	24	3	24/27	**88.88%**	**26.85**
80	23	2	23/25	**92%**	**25.05**
90	23	1	23/24	**95.8%**	**26.85**

Table 4.5 Detection Rate in frames with “Pedestrian_YOLOv4”

Frame	Detected	Not Detected	Total	Percentage	FPS (Tesla K80)
10	8	19	8/27	29%	20.59
20	10	16	10/26	38.4%	21.8
30	11	16	11/27	40.74%	26.64
40	10	20	20/30	66.66%	21.46
50	12	13	12/25	48%	20.64
60	10	17	10/27	37.03%	20.62
70	12	15	12/27	44.44%	21.86
80	10	15	10/25	40%	20.64
90	9	15	9/24	37.5%	21.86

Table 4.6 detection Rate in frames with YOLOv4.

As we notice in Table 4.5 and 4.6 our proposed method gave far better results in terms of detection rate and better FPS rate (4~6 FPS gain) than the old method in extremely low light conditions (night time), where there's less information to process.

D. Tracking Performance

we count only the perfect tracked or partly tracked with more than 30 frames. And only the pedestrian who didn't left the scene (from frame 1 to 90) based on MOT challenge video that been processed to be visibly challenging (Figure 4.16).

Method	Track count (From frame 0 to 90)	Tracked more than 30 frames	Total	Ground truth Count
YOLOv4-DeepSORT	7	3	10	31
P_YOLOv4-DeepSORT (OURS)	**13**	**8**	**21**	31

Table 4.7 Tracks comparison in extremely low light conditions

we can see in Table 4.7; we can deduce two things:

- the detection in every frame plays a major role for better and consisting tracking performance.
- Original YOLOv4 model finds hard time in detecting the class Pedestrians during extremely low light conditions.

4.3 – limitations

We observe that the results were promising specially in low light conditions, where we can see the "YOLOv4" finds hard time to find and detect people at night, in the same time "Pedestrian_YOLOv4" did detection more accurately duo to the architecture that was person specified and keeping it as fast

as possible, and we can notice in the experiments, there were more detections of pedestrian generated in a scene specially in low light conditions, and therefore more information (bounding boxes) for DeepSORT to follow and do the data association between each detection in every frame, but even though there's still some limitation to discover during our experimentations:

1. The offline tracking algorithms still shows better results than online tracking algorithms, duo to their peaking to future frames and knowing where the object locations in the future, but unfortunately these algorithms can't be applied to a live scenario as we wish to achieve in this master thesis project.
2. The ID switch, because in some of the experimentation showed that Sometimes When two Pedestrians or more, get really close the DeepSORT algorithm fails to assign the right tracks to the right detection.
3. This pipeline depends on the quality of detections and interference time, in some cases where the detector fails in detecting a pedestrian in a given frame or number of frames duo to an occlusion (by some wall for example or Pedestrians occlude another) the detection here obviously will be missed and the update of on a track will be missed too, therefore DeepSORT fails to re-identify the object that stays unobserved (Undetected) for A_{max} age and consider it as lost, and if the object get detected again the Algorithm DeepSORT consider it as a lost track.
4. Results where the camera is moving is not as good as the results with the fixed camera duo to the lack of ego motion information in the motion prediction calculation.

4.4 – Conclusion

We observed that YOLOv4 were trained to know more than one object class (Person, cat, dog…etc), showed poor results specially in terms of Pedestrian detection class, in low or extremely low light conditions and in Grayscale inputs (less information), we corrected this by retraining and modifying the YOLOv4's CNN architecture to output one class (which is the pedestrian in our case of interest) by collecting a new dataset that was person specified. And re-implement the architecture in a GPU environment for faster calculation and FPS, and therefore, our model "Pedestrian_YOLOv4" worked better with DeepSORT in tracking pedestrians, and showed better results in in terms of speed, detection and tracking performance specially in extremely low light conditions and Grayscale inputs.

General Conclusion

Real time object tracking is a challenging task for computer vision and in computer science in general, where it has so many utilities and wide range of use (like security cameras, self-driving cars…etc), where the accuracy and speed are two major factors and we can't separate one from another specially in a live scenario, that requires a quick response time (like a car that immediately stops, where it detects a pedestrian or another car on the road to avoid accidents). This kind of problems need faster calculation time without losing the accuracy, we saw that real-time tracking can be divided in two parts **offline tracking** and **online tracking**:

- The online tracking only counts on the information given in a frame and the previous one (like humans)
- The offline tracking peaks to future frames for better trajectories estimation, which is not useful in live scenarios where a reaction is not required

State-of-the-Art online algorithms now are breaking records of the offline tracking methods, which gives approximately the same results but this comes with more computational cost (drop in speed).

In order to build a robust and fast pedestrian tracking for live scenarios, the accuracy and speed are required, so we have to choose among State-of-the-Art algorithms that has an excellent balance between the accuracy and speed, we found the Simple Online real-time tracker (SORT).

SORT is a detection-based tracker (DBT family) created in 2016, which is strong competitor to other online tracking frameworks in terms of accuracy, with the perks of being the fastest among them, duo to its simple architecture that depends on Faster RCNN detections, the Kalman Filtering for motion prediction and a Hungarian algorithm that associate detections from frame to another by giving it the same ID based on their IOU match.

The results were promising but showed a huge amount of the ID switch phenomena, where it assigns the wrong detection to the wrong tracks because it just assigns boxes without knowing what's inside.

A year later in mars 2017 an extension of this algorithms came out called DeepSORT, is uses a CNN based feature Extractor, where it extracts features from the detected bounding boxes and use it as one of the scores for associating a track to a detection with a min cost matrix that finds the minimum distance between their feature vectors.

Besides feature Extraction, they also updated their data association process, which now does not only counts on the IOU Match but also on the Mahalanobis distance that takes care of the frame-by-frame data association, and a cosine distance between 128 dimension vector of track and a given detection, with a predefined threshold to compete in the matching cascade phase, where it assigns tracks' prediction to the correct detection with the Hungarian algorithm that solves optimally a linear problem between set of tracks and detections, and once again the DeepSORT algorithm compete for state-of-the-art algorithms with a little trade off with speed.

DeepSORT philosophy is to keep the complex calculations to the detector, we now know that it relays on Faster RCNN in the detection step.

In April 2020 the state-of-the-art object detector YOLOv4 came out to break all previous records in terms of speed and accuracy even better than Faster RCNN. YOLOv4 is a CNN based multi-object detector who has the ability to detect bounding boxes with multi class classification.

The idea here is to replace DeepSORT detector, instead of counting on Faster RCNN which works on two forward passes, by YOLOv4 who works with one forward pass (Time gain).

The results where good but not satisfactory in terms of detecting the pedestrian class and therefore in pedestrian tracking, especially in night times and extremely low light situations, and Grayscale inputs (which a lot of security camera are) and all those conditions are important in pedestrian tracking.

And therefore, we have improved the YOLOv4 architecture by making it only focuses on detecting pedestrian with keeping its speed (to serve this master thesis purpose), and feed the DeepSORT algorithm and do the data association.

So, we proposed a robust and fast method "Pedestrian_YOLOv4-DeepSORT" to track pedestrians in hard visual conditions including Grayscale inputs with extremely low light conditions.

This master thesis focuses on the importance of the detection part in the tracking overall performance by proposing a better method in terms of pedestrian tracking that equally gives importance to the accuracy and speed.

Future objectives:

- Add more scores to the Hungarian algorithm that associate detection to tracks by improving Re-ID CNN model (the feature extractor).
- Detecting and tracking more than 100 people in one frame.
- Add more classification in the Pedestrian class.
- Making it faster for Mobile uses.

Bibliography

[1] Paul Voigtlaender, Michael Krause, Aljosa Osep, Jonathon Luiten, Berin Balachandar Gnana Sekar, Andreas Geiger, and Bastian Leibe. Mots: Multi-object tracking and segmentation. arXiv preprint arXiv:1902.03604, 2019.

[2] Arthur L Samuel. Some studies in machine learning using the game of checkers. IBM Journal of research and development, 44(1.2):206–226, 2000.

[3] Mohamed Chaabane, Peter Zhang, J. Ross Beveridge, Stephen O'Hara CenterTrack paperwork, Feb 31, 2021 arXiv:2102.02267v1

[4] Juan Du Understanding of Object Detection Based on CNN Family and YOLO (pp 4-5), 2018

[5] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016

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

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

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

[9] L. Cun et al., "Handwritten Digit Recognition with a Back-Propagation Network,"

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

[11] L. Cun et al., "Handwritten Digit Recognition with a Back-Propagation Network," 1990.

[12] Bo Wu and Ram Nevatia. Tracking of multiple, partially occluded humans based on static body part detection. In 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06), volume 1, pages 951–958. IEEE, 2006.

[13] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. Journal on Image and Video Processing, 2008:1, 2008.

[14] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara, and Carlo Tomasi. Performance measures and a data set for multi-target, multi-camera tracking. In European Conference on Computer Vision, pages 17–35. Springer, 2016.

[15] Rainer Stiefelhagen and John Garofolo. Multimodal Technologies for Perception of Humans: First International Evaluation Workshop on Classification of Events, Activities and Relationships, CLEAR 2006, Southampton, UK, April 6-7, 2006, Revised Selected Papers, volume 4122. Springer, 2007.

[16] Rainer Stiefelhagen, Rachel Bowers, and Jonathan Fiscus. Multimodal Technologies for Perception of Humans: International Evaluation Workshops CLEAR 2007 and RT 2007, Baltimore, MD, USA, May 8-11, 2007, Revised Selected Papers, volume 4625. Springer, 2008.

[17] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. arXiv preprint arXiv:1504.01942, 2015.

[18] 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.

[19] P. Dollar, C. Wojek, B. Schiele, and P. Perona, “Pedestrian detection: An evaluation of the state of the art,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 34, pp. 743–761, April 2012

[20]/4 M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The pascal visual object classes (voc) challenge,” International Journal of Computer Vision, vol. 88, pp. 303–338, Jun 2010.

[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollar, and C. L. Zitnick, “Microsoft coco: Common objects in ´ context,” in Computer Vision – ECCV 2014 (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 740–755, Springer International Publishing, 2014

[22] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. ernstein, A. C. Berg, and L. Fei-Fei, “Imagenet large scale visual recognition challenge,” International Journal of Computer Vision, vol. 115, pp. 211–252, Dec 2015.

[23] A. Kuznetsova, H. Rom, N. Alldrin, J. Uijlings, I. Krasin, J. Pont-Tuset, S. Kamali, S. Popov, M. Malloci, T. Duerig, and V. Ferrari, “The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale,” arXiv:1811.00982, 2018.

[23] P. Dendorfer et al., “MOT20: A benchmark for multi object tracking in crowded scenes,” Mar. 2020.

[24] Gioele Ciaparrone et al., “DEEP LEARNING IN VIDEO MULTI-OBJECT TRACKING” November 20,2018

[25] Laura Leal-Taixé, Anton Milan, Ian Reid, Stefan Roth, and Konrad Schindler. Motchallenge 2015: Towards a benchmark for multi-target tracking. arXiv preprint arXiv:1504.01942, 2015.

[26] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. IEEE transactions on pattern analysis and machine intelligence, 36(8):1532–1545, 2014.

[27] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9):1627–1645, 2009.

[28] Ross B. Girshick, Pedro F. Felzenszwalb, and David McAllester. Discriminatively trained deformable part models, release 5. http://people.cs.uchicago.edu/~rbg/latent-release5/, 2012.

[29] Fan Yang, Wongun Choi, and Yuanqing Lin. Exploit all the layers: Fast and accurate cnn object detector with scale dependent pooling and cascaded rejection classifiers. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2129–2137, 2016.

[30] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems, pages 91–99, 2015.

[31] Patrick Dendorfer, Hamid Rezatofighi, Anton Milan, Javen Shi, Daniel Cremers, Ian Reid, Stefan Roth, Konrad Schindler, and Laura Leal-Taixe. Cvpr19 tracking and detection challenge: How crowded can it get?, 2019.

[32] Andreas Geiger, Philip Lenz, and Raquel Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. In 2012 IEEE Conference on Computer Vision and Pattern Recognition, pages 3354–3361. IEEE, 2012.

[33] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. The International Journal of Robotics Research, 32(11):1231–1237, 2013.

[34] Xiaoyu Wang, Ming Yang, Shenghuo Zhu, and Yuanqing Lin. Regionlets for generic object detection. In Proceedings of the IEEE international conference on computer vision, pages 17–24, 2013.

[35] Hieu Tat Nguyen and Arnold WM Smeulders. Fast occluded object tracking by a robust appearance filter. IEEE Transactions on Pattern Analysis and Machine Intelligence, 26(8):1099–1104, 2004.

[36] Dorin Comaniciu, Visvanathan Ramesh, and Peter Meer. Real-time tracking of nonrigid objects using mean shift. In Computer Vision and Pattern Recognition, 2000. Proceedings. IEEE Conference on, volume 2, pages 142–149. IEEE, 2000.

[37] Simon Baker and Iain Matthews. Lucas-kanade 20 years on: A unifying framework. International journal of computer vision, 56(3):221–255, 2004

[38] Hieu T Nguyen and Arnold WM Smeulders. Robust tracking using foregroundbackground texture discrimination. International Journal of Computer Vision, 69(3):277–293, 2006

[39] Piotr Dollár, Ron Appel, Serge Belongie, and Pietro Perona. Fast feature pyramids for object detection. IEEE transactions on pattern analysis and machine intelligence, 36(8):1532–1545, 2014.

[40] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. IEEE transactions on pattern analysis and machine intelligence, 32(9):1627–1645, 2009.

[41] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, June 2016.

[42] T. Lin, P. Dollr, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 936– 944, July 2017.

[43] Z. Li, C. Peng, G. Yu, X. Zhang, Y. Deng, and J. Sun, "Detnet: A backbone network for object detection," arXiv preprint arXiv:1804.06215, 2018.

[44] Joseph Redmon Ali Farhadi "YOLOv3: An Incremental Improvement" 8 April, 2018

[45] K. He, G. Gkioxari, P. Dollr, and R. Girshick, "Mask r-cnn," in 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2980– 2988, Oct 2017.

[46] W. Choi, "Near-online multi-target tracking with aggregated local flow descriptor," in ICCV, 2015, pp. 3029– 3037.

[47] X. Zhang, X. Zhou, M. Lin, and J. Sun, "Shufflenet: An extremely efficient convolutional neural network for mobile devices," in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pp. 6848–6856, 2018.

[48] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587, June 2014

[49] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 779–788, June 2016.

[50] R. Girshick, J. Donahue, T. Darrell, and J. Malik, “Rich feature hierarchies for accurate object detection and semantic segmentation,” in 2014 IEEE Conference on Computer Vision and Pattern Recognition, pp. 580–587, June 2014

[51] Seungkwan Lee, Suha Kwak, and Minsu Cho “Universal Bounding Box Regression and Its Applications”

[52] R. Girshick, “Fast r-cnn,” in 2015 IEEE International Conference on Computer Vision (ICCV), pp. 1440–1448, Dec 2015.

[53] J.R.R. Uijlings, K.E.A. van de Sande, T. Gevers, and A.W.M. Smeulders “Selective Search for Object Recognition”, September 2013

[54] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 39, pp. 1137–1149, June 2017.

[55] B. Yang and R. Nevatia, “Online learned discriminative partbased appearance models for multi-human tracking,” in Proc. Eur. Conf. Comput. Vis., 2012, pp. 484–498.

[56] W. Hu, X. Li, W. Luo, X. Zhang, S. Maybank, and Z. Zhang, “Single and multiple object tracking using log-euclidean riemannian subspace and block-division appearance model,” IEEE Trans. Pattern Anal. Mach. Intel., vol. 34, no. 12, pp. 2420–2440, Dec. 2012.

[57] L. Zhang and L. van der Maaten, “Structure preserving object tracking,” in Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit., 2013, pp. 1838–1845.

[58] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos, Ben Upcroft “SIMPLE ONLINE AND REALTIME TRACKING” version 2 July, 2017

[59] Nicolai Wojke, Alex Bewley, Dietrich Paulus “SIMPLE ONLINE AND REALTIME TRACKING WITH A DEEP ASSOCIATION METRIC” march 2017

[60] S. Ren, K. He, R. Girshick, and J. Sun, “Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks,” in Advances in Neural Information Processing Systems, 2015

[61] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao and Tae-Kyun Kim “Multiple Object Tracking: A Literature Review” 22 may 2017

[62] R. Kalman, “A New Approach to Linear Filtering and Prediction Problems,” Journal of Basic Engineering, vol. 82, no. Series D, pp. 35–45, 1960.

[63] H. W. Kuhn, “The Hungarian method for the assignment problem,” Naval Research Logistics Quarterly, vol. 2, pp. 83–97, 1955.

[64] L. Leal-Taix´e, A. Milan, I. Reid, S. Roth, and K. Schindler, “MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking,” arXiv preprint, 2015.

[65] Will Koehrsen “Transfer leaning with convolution neural networks” NOV 29th, 2018 https://towardsdatascience.com/transfer-learning-with-convolutional-neural-networks-in-pytorch-dd09190245ce

[66] Sangdoo Yun1et al “CutMix: Regularization Strategy to Train Strong Classifiers with Localizable Features” 7 august 2019

[67] Vishal Mandal and Yaw Adu-Gyamfi, “Object Detection and Tracking Algorithms for Vehicle Counting: A Comparative Analysis”, November,2020

[68] Duan K, Bai S, Xie L, Qi H, Huang Q, Tian Q (2019) Centernet: keypoint triplets for object detection. In: Proceedings of the IEEE International Conference on Computer Vision, pp 6569–6578

[69] Wu Y, Kirillov A, Massa F, Lo W-Y, Girshick R (2019) Detectron2

[70] Tan M, Pang R, Le QV (2020) Efcientdet: scalable and efcient object detection. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pp 10781–10790

[71] Bochinski E, Senst T, Sikora T (2018) Extending IOU based multiobject tracking by visual information. In: 2018 15th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS), IEEE, pp 1–6

[72] Erik Bochinsk et al, “Extending IOU Based Multi-Object Tracking by Visual Information”, November 2018

[73] “Real-Time Object Detection on COCO” https://paperswithcode.com/sota/real-time-object-detection-on-coco?metric=FPS

[74] Aleksey Bochkovskiy “YOLOv4 — the most accurate real-time neural network on MS COCO dataset” may 21, 2020. https://alexeyab84.medium.com/yolov4-the-most-accurate-real-time-neural-network-on-ms-coco-dataset-73adfd3602fe

[75] Alexey Bochkovskiy, Chien-Yao Wang, Hong-Yuan Mark Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection”, 23 Apr 2020

[76] Joseph Redmon, Ali Ferhadi “YOLOv3: An Incremental Improvement”, April 2018

[77] Riccardo Bonetto, Vincent Latzko, in Computing in Communication Networks, 2020, https://www.sciencedirect.com/topics/computer-science/squared-error-loss

[78] Speech and Language Processing. Daniel Jurafsky & James H. Martin, “chapter 5 logistic Regression” December 30, 2020.

[79] Usha Ruby, Vamsidhar Yendapalli, “Binary cross entropy with deep learning technique for Image classification”, August 2020

[80] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 2117–2125, 2017

[81] J. Redmon and A. Farhadi. Yolo9000: Better, faster, stronger.In Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on, pages 6517–6525. IEEE, 2017

[82] Rasmus Rothe, Matthieu Guillaumin, Luc Van Gool, “Non-Maximum Suppression for Object Detection by Passing Messages between Windows”, April 2015

[83] Golnaz Ghiasi et al “DropBlock: A regularization method for convolutional networks” 30 october 2018

[84] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014

[85] Tsung-Yi Lin, Piotr Dollar, Ross Girshick, Kaiming He, ´ Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2117–2125, 2017

[86] Shu Liu, Lu Qi, Haifang Qin, Jianping Shi, and Jiaya Jia. Path aggregation network for instance segmentation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 8759–8768, 2018.

[87] Chien-Yao Wang, Hong-Yuan Mark Liao, Yueh-Hua Wu, Ping-Yang Chen, Jun-Wei Hsieh, and I-Hau Yeh. CSPNet: A new backbone that can enhance learning capability of cnn. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshop (CVPR Workshop), 2020

[88] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI), 37(9):1904–1916, 2015

[89] Wenhan Luo “Multiple Object Tracking: A Literature Review” 22 may 2017.

[90] L. Zheng, Z. Bie, Y. Sun, J. Wang, C. Su, S. Wang, and Q. Tian, “MARS: A video benchmark for large-scale person re-identification,” in ECCV, 2016.

[91] Joseph Redmon’s website creator of YOLO https://pjreddie.com/darknet/1

[92] Chris Harris and Mike Stephens. A combined corner and edge detector. In Alvey vision conference, volume 15, pages 10–5244. Manchester, UK, 1988.

[93] Jianbo Shi et al. Good features to track. In Computer Vision and Pattern Recognition, 1994. Proceedings CVPR’94., 1994 IEEE Computer Society Conference on, pages 593–600. IEEE, 1994.

[94] David Held, Sebastian Thrun, and Silvio Savarese. Learning to track at 100 fps with deep regression networks. In European Conference on Computer Vision, pages 749–765. Springer, 2016

[95] Boris Babenko, Ming-Hsuan Yang, and Serge Belongie. Visual tracking with online multiple instance learning. In Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on, pages 983–990. IEEE, 2009

[96] Jonathan Hui “mean Average Precision for object detection”, mars 7th, 2018 “https://jonathan-hui.medium.com”

[97] Wikipedia Precision and recall https://en.wikipedia.org/wiki/Precision_and_recall

[98] Terrance DeVries and Graham W. Taylor, “Improved Regularization of Convolutional Neural Networks with Cutout”, 29 November 2017

[99] Tesla Andrej Karpathy in CVPR, June 14-19 2020: Scalability in Autonomous Driving Workshop