Should every stage of track-by-detection utilize deep learning?

Pro Gradu
University of Turku
Department of Computing
Computer Science
July 2021
Juho Kuusinen

The originality of this thesis has been checked in accordance with the University of Turku quality assurance system using the Turnitin OriginalityCheck service.

University of Turku
Department of Computing

JUHO KUUSINEN: Should every stage of track-by-detection utilize deep learning?

Pro Gradu, 69 pages.
Computer Science
July 2021

The neural network-based solutions are becoming more and more popular because of their ability to solve problems, which could not be solved before. This has also led people to utilize neural networks to solve problem, where more classical methods could be utilized. This work tries to solve if the usage of neural networks in track-by-detection paradigm gives the system an advance over system with classical methods when tracking pedestrians. Track-by-detection is a two-module system, where the first module extracts detections from an input image. Detections are fed to the second module, which associates detections with unique identifier and tries to track identified objects through a sequence of concurrent images.

The hypothesis of this work is that both modules in track-by-detection can be replaced with solutions without a neural network. Research was performed for both modules separately, be because the object detection can be evaluated without the second module, which can be evaluated with precalculated detections. Research about object detection was done as a literature review. Different tracking algorithms were evaluated using MOTChallenge's data set. According to the results from the literature review, object detection cannot be replaced with classical methods. The results about tracking shows that tracking can be done well without neural networks.

Results of this work shows that neural network-based solutions are justified to be used in the first module of track-by-detection. The second module can be neural network-based, but this required more resources to get working well. Many of the more classical methods can be swapped easily in the track-by-detection to find which methods works best in the current use-case. According to the results, neural network-based trackers do not bring enough benefits for real-time tracking to be used over classical methods.

Keywords: Object detection, Tracking-by-detection, Tracking, Pedestrians, Deep Learning, Convolutional neural network, YOLO, HOG, SORT, MOTChallenge, Deep SORT, Kalman filter, Real-time tracking

Turun yliopisto
Tietotekniikan laitos

JUHO KUUSINEN: Pitäisikö jokaisen havaintopohjaisen seurannan vaiheen hyödyntää syväoppimista?

Pro Gradu, 69 sivua
Tietojenkäsittelytiede
Heinäkuu 2021

Neuroverkkopohjaisten ratkaisujen suosio on jatkuvassa kasvussa, koska niiden avulla kyetään ratkaisemaan ongelmia, joita ei ole ennen voinut ratkaista. Tämä on kuitenkin johtanut siihen, että neuroverkkoja käytetään ratkaisemaan ongelmia, joihin perinteiset menetelmät toimisivat hyvin. Tässä työssä pyritään selvittämään, onko neuroverkkojen hyödyntäminen jalankulkijoiden havaintopohjaisessa seurannassa tarpeen. Havaintopohjainen seuranta (Tracking-by-Detection) on paradigma, joka muodostuu kahdesta moduulista. Ensimmäinen moduuli hoitaa kohteen tunnistuksen annetusta kuvasta ja lähettää havainnot seuraavalle moduulille. Toisen moduulin työ on luoda uniikkeja tunnisteita havainnoille ja yhdistää peräkkäisten kuvien havainnot toisiinsa.

Tämän työn hypoteesi on, että havaintopohjaisen seurannan molemmat moduulit voidaan korvata klassisilla menetelmillä, jotka eivät hyödynnä neuroverkkoja. Molempia moduuleja tutkittiin erikseen, koska molempia moduuleja voidaan arvioida ilman toista. Ensimmäinen moduuli arvioitiin kirjallisuuskatsauksena ja toinen moduuli arvioitiin hyödyntäen MOTChallengen arviointikriteerejä. Kirjallisuuskatsauksen tuloksien perusteella kohteen tunnistusta ei voida korvata järkevästi klassisilla menetelmillä. Havaintojen seurantaan käytettävä moduuli voidaan korvata menetelmillä, jotka eivät käytä neuroverkkoja.

Työn tuloksien mukaan neuroverkkopohjaiset ratkaisut ovat oikeutettuja käytettäväksi havaintopohjaisen paradigman ensimmäisessä moduulissa. Toisessa moduulissa neuroverkkopohjaisia ratkaisuja voidaan hyödyntää, mutta tällöin ratkaisun luominen vaatii enemmän resursseja kehitysvaiheessa. Klassisilla menetelmillä voidaan helposti ja nopeasti kokeilla eri menetelmiä löytääksemme parhaimman mahdollisimman menetelmän ratkaistavalle käyttötapaukselle. Tuloksien mukaan neuroverkkopohjaiset seurantamenetelmät eivät tuo tarpeeksi hyötyä reaaliaikaisessa seurannassa, jotta niiden käyttäminen klassisten menetelmien sijaan olisi oikeutettua.

Avainsanat: Kohteen tunnistus, Havaintopohjainen seuranta, Seuranta, Jalankulkijat, Syväoppiminen, Konvoluutioverkko, YOLO, HOG, SORT, MOTChallenge, Deep SORT, Kalman suodatin, Reaaliaikainen seuranta

Contents

List of Figures i

List of Tables iii

List Of Acronyms iv

1 Introduction 1

2 Artificial neural network 4

2.1 Introduction 4

2.2 Artificial neuron 4

2.3 Artificial neuron activation 5

2.4 Neural network layers 9

3 Convolutional neural network 11

3.1 Introduction 11

3.2 Convolution 12

3.3 Pooling 14

4 Object Recognition 16

4.1 Introduction 16

4.2 Why is the object recognition important? 16

4.3 Image classification 17

4.4 Object detection 17
4.5 Semantic Segmentation 19
4.6 Instance segmentation 19
4.7 Cleaning detections 20
4.8 Evaluation of object detectors 22

5 The Multiple Object Tracking Benchmark 25
5.1 Introduction 25
5.2 MOTChallenge data format 26
5.3 Evaluation 28

6 Object Detector 31
6.1 Introduction 31
6.2 Histogram of Oriented Gradients and Support Vector Machine 33
6.3 Region Based Convolutional Neural Network 36
6.4 You Only Look Once 39

7 Data Association 42
7.1 Introduction 42
7.2 Euclidean distance 43
7.3 Kalman filter 48
7.4 Simple Online and Realtime Tracking 50
7.5 Sort with Deep Association Metric 52

8 Results 54
8.1 Object detectors 54
8.2 Tracking 55

9 Conclusion 60

References 62

Appendices

A Naive object tracker using Euclidean distance A-1

B Naive object tracker using Kalman filter B-1

List of Figures

2.1 Mathematical model for neuron by McCulloch & Pitts 5
2.2 Visualization of data where decision boundary with linear solution is enough 6
2.3 Visualization of data where linear decision boundary does not work . . . 7
2.4 Sigmoid function with weight changing 8
2.5 Sigmoid function with bias changing 8

3.1 Amount of publications according to the search word *Convolutional neural network* in Google Scholar. 12
3.2 Sobel kernels for X and Y 13
3.3 Adding padding to the matrix M 14
3.4 Sobel edge detection. 14
3.5 Max pooling with a kernel of $2x2$ and stride of 2. 15

4.1 Face detection done with Haar Cascades 18
4.2 Semantic segmentation done to a picture. 19
4.3 Instance segmentation where only pedestrians are detected 20
4.4 With T_{iou} of 0.5, 10 true positive detections were lost after NMS 21

5.1 Visualization of values for PETS09-S2L1 frame 14 27

6.1 Visualization of how the histogram from regions could be visialized . . . 34
6.2 Visualization of SVM splitting data points 35

6.3 Visualization of different parts of selective search 37

7.1 Visualization of identifiers switching for Euclidean distance 44
7.2 Euclidean distance tracking with visualization of maximum distance . . . 45
7.3 Hungarian algorithm steps 48
7.4 Kalman filter predicting objects location with constant speed from left to right 50
7.5 Kalman filter predicting object that is moving in curve moving from right to left 50

List of Tables

7.1 Deep SORT's CNN architecture 53

8.1 MOT20 training sequences' data 55

8.2 Trackers' results 55

8.3 Results for Euclidean distance based tracker 56

8.4 Results for Kalman filter based tracker 57

8.5 Results for SORT 58

8.6 Results for Deep SORT 59

List Of Acronyms

ANN Artificial Neural Network

CLS Box-classification layer

CNN Convolutional neural network

DNN Deep neural network

HOG Histogram of Oriented Gradients

IDF1 Identification F1 Score

IDFN False Negative ID

IDFP False Positive ID

IDP Identification Precision

IDR Identification Recall

IDTP True Positive ID

IoU Intersection over union

mAP mean Average Precision

ML Mostly lost

MOT Multiple Object Tracking

MOTA Multi-Object Tracking Accuracy

MOTP Multiple Object Tracking Precision

MT Mostly tracked

NMS Non-Maximum Suppression

PT Partially tracked

R-CNN Region Based Convolutional Neural Network

REG Box-regression layer

ReLU Rectified Linear Unit

RoI Region of Interest

RPN Region Proposal Network

SORT Simple Online and Realtime Tracking

SVM Support Vector Machine

TBD Tracking-by-Detection

YOLO You Only Look Once

Chapter 1

Introduction

Solutions utilizing neural networks are becoming increasingly popular. Many start-ups focus on purely neural network-based solutions, namely deep learning for solving problems. Gathering data and processing it for training and testing different deep learning methods is very time and resource consuming. Many of these provided solutions are focusing on analysing images or sequences of images. The goal is usually to extract different objects from them and perform post-processing fitting for the specific problem at hand.

In this work, I will focus on associating detections between concurrent frames. This associating process will be inspected from the point of a paradigm called Tracking-by-Detection. Tracking-by-Detection is commonly used paradigm when creating any kind of tracking application which utilizes images. This work will be focusing on paradigm capability for Multiple Object Tracking and I will exclude its other tracking category Single Object Tracking.

Research question for this work is, that is it possible to change parts from Tracking-by-Detection to be not related to neural networks? More specifically if it is possible to replace any of the parts when detecting and tracking pedestrians. Most of the de facto neural network-based solutions are utilizing massive networks and using highly parallel calculations by using a Graphics Processing Unit (GPU). I think, that using neural network-based solutions for everything is not efficient, and simpler solutions could do as

well or almost as well as solutions utilizing neural networks. This is significant also because of the combination of virtual currency mining becoming increasingly popular and the COVID-19, there is right now shortage of GPUs. This shortage has made GPUs much pricier and harder to get. Because of this situation, I think it is important to try to discover use-cases where using neural network is unnecessary.

Comparison of different object detections will be performed as a review of different research papers. This can be done, because there are massive amounts of already published papers about how different object detectors compare for each other. Object detectors that will be inspected in this work are two neural network-based solutions: You Only Look Once (YOLO) v3 and Region Based Convolutional Neural Network (R-CNN). To represent implementation that is not based on neural network, I chose Histogram of Oriented Gradients (HOG) with Support Vector Machine (SVM), because HOG was designed for pedestrian detection.

For testing different data association methods to connect detections between frames, I will use data set provided by The Multiple Object Tracking Benchmark as MOTChallenge [1]. These data association methods are called as tracking algorithms. In this work, four tracking algorithms are compared to each other. Two of them are very simple and the other two were state-of-the-art algorithms. First two are trackers utilizing just simple mathematical concepts: Euclidean distance and Kalman filter. Two other algorithms are called Simple Online and Realtime Tracking (SORT) and Sort with Deep Association Metric (Deep SORT). Only Deep SORT uses neural network from selected trackers.

In this work, we will first address what is a neural network and how does it work in chapter 2. After that we will discuss about Convolutional neural network in chapter 3, which is a deep learning method and a type of neural network. In its sub-chapters, we familiarize ourselves how does the CNN differ from other neural networks. In chapter 4, we talk about object recognition, one of the main tasks for CNN. In that chapter we discuss what is object recognition, what kind of tasks it contains and how to post-process

the outputs from object recognition. Main data set used in this work will be introduced in chapter 5. Different object detectors and their main ideas are discussed in chapter's 6 sub-chapters and the paradigm Tracking-by-Detection is introduced and described in chapter 6.1. Algorithms used for data associations will be discussed in chapter's 7 sub-chapters. Finally, this work's results are shown in chapter 8, where we will go through results from literature and tracking evaluation results. After examining results, we will discuss about the results in chapter 9, where we will make the conclusion.

Chapter 2

Artificial neural network

2.1 Introduction

When talking about object detection, first we need to understand what Artificial Neural Network (ANN) is. Neural networks are collections of different layers created by neurons. Layers are combined by different methods to define how the information flow from layer to layer. Before going to the technical details, let's try to grasp the general idea of neural networks. Artificial Neural Network represents an artificial (also called as *Neural Network*) version of a biological neural network. In a biological neural network, neurons communicate with short electrical signals and one neuron can have thousands of connections to other neurons. Artificial neural network is a collection of artificial neurons which are capable for sending signals and receiving and processing received signals. [2]

2.2 Artificial neuron

Artificial neuron can be visualised as a processor that performs simple mathematical operation. We are not interested in how the biological neurons works, so instead of that we are going to focus on a mathematical model for neuron. In 1943 McCulloch and Pitts introduced simple mathematical model for neuron. [3]

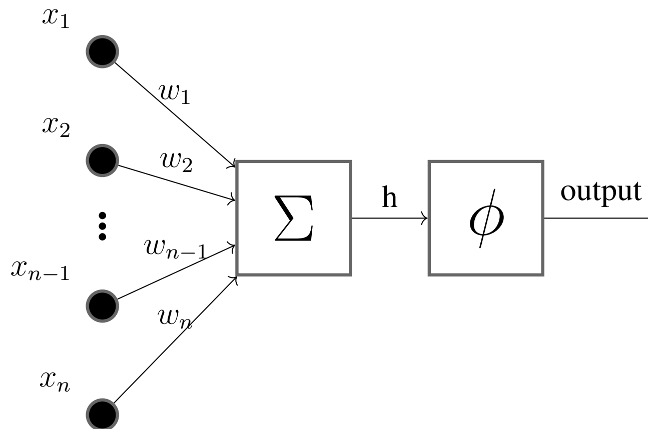


Figure 2.1: Mathematical model for neuron by McCulloch & Pitts [3]

In Figure 2.1 we have a model of the mathematical neuron designed by McCulloch and Pitts. The shown model can be split into three parts: inputs, adder and activation function. Inputs are labelled as x_i and for each input there exists a weight w_i. These variables are shown in Figure 2.1 on the left side. The adder represents real cells membrane, so it stores given signals, but in this model, it works as a simple sum-up function (Σ). In Equation 2.1 [3] we can see how the adder works. It sums up the products of x_i and w_i. The last part of this model is the activation function (also known as *Transfer function*) which purpose is to decide the final output of the artificial neuron. We will cover activation function in Section 2.3.

$$h = \sum_{i=1}^{n} x_i w_i \tag{2.1}$$

2.3 Artificial neuron activation

We could determine a neuron's output with just a sum of bias and the product of input and weight. A problem with this approach is that it is a linear function. For example, this allows our decision boundary to be only a straight line in a two-dimensional space. Decision boundary is a boundary that divides decision space for the algorithm [4]. For

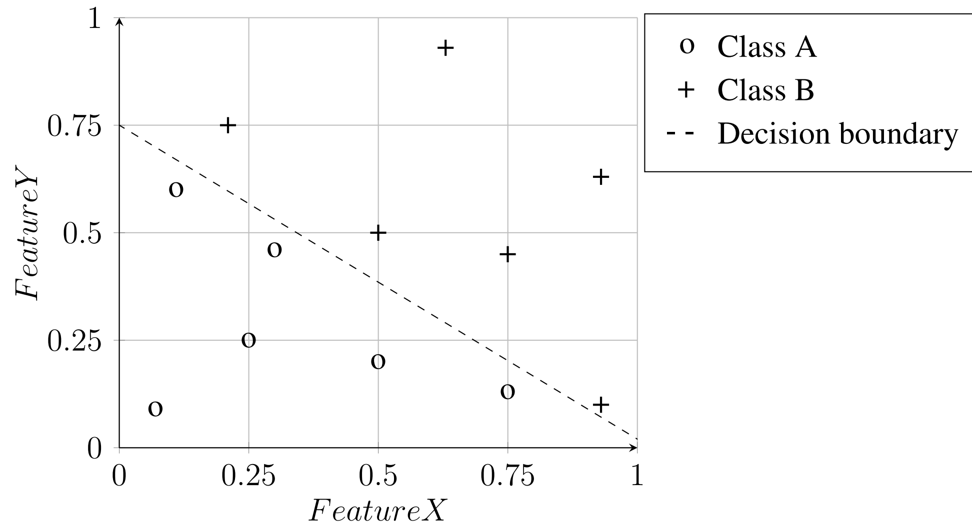


Figure 2.2: Visualization of data where decision boundary with linear solution is enough

input of two features, decision boundary is easy to plot. In Figure 2.2, we can see how linear function divides to two categories easily, but if the data is even a little bit more complicated, we cannot solve it with a linear function as we can see in Figure 2.3.

To solve non-linear problems, we need to add non-linearity to our function. This is where activation functions are coming to play. The simplest activation function is the one used in McCulloch's and Pitts artificial neuron model. This activation function is called a Heaviside step function [5]. Heaviside step-function is a discontinuous, function and it can be used to produce an output of zero (0) or one (1). Let's indicate the neuron's threshold with θ. If the h is larger than θ, output will be one. If it is not, the output will be zero.

$$o = g(h) = \begin{cases} 1, & h > \theta \\ 0, & h \leq \theta \end{cases} \tag{2.2}$$

Even if Heaviside step-function is used in the original model as an activation function, it is not usually used as an activation function.

One used activation function is called *Sigmoid function*[2]. On the contrary to the Heaviside step-function, sigmoid function's output is a value between zero (0) and (1), where in the Heaviside step-function, the output is either zero (0) or one (1). With Sigmoid

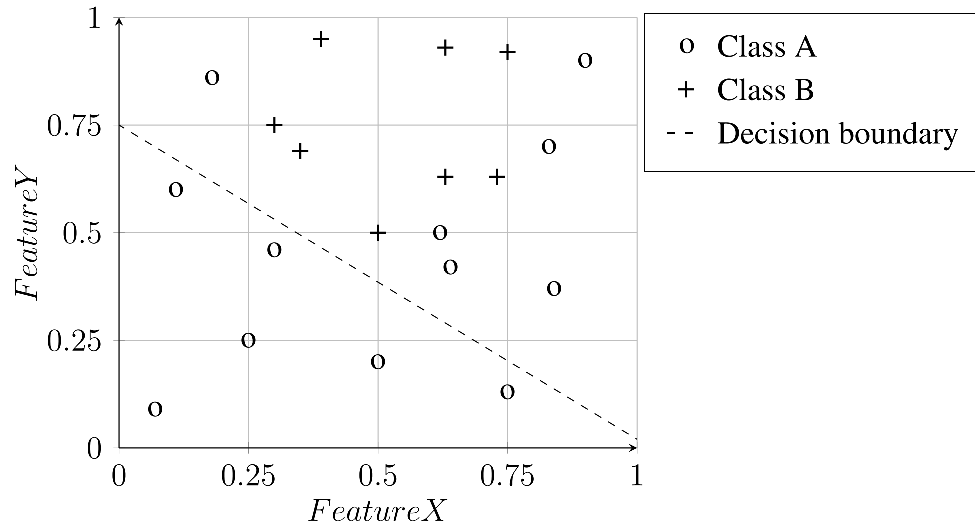


Figure 2.3: Visualization of data where linear decision boundary does not work

function, we can map any value between zero (0) and one (1) and the value of 0.5 will given if the x is 0 for the equation 2.3[2].

$$S(x) = \frac{1}{1 + e^{-x}} \tag{2.3}$$

Earlier in chapter 2.2, we discussed little about weights. When we add weights to the sigmoid function, we get the equation 2.4. With the weight parameter, we can modify the sigmoid function in different ways. We know that if weight w is one (1), it does not affect the function's output in any way. If we set the weight to be negative one (-1), we inverse the values as we can see in a Figure 2.4. As mentioned before in this chapter, if the power of e's is zero (0) we always get the same result of 0.5 as shown in figure 2.4 and 2.5.

$$S(x) = \frac{1}{1 + e^{-x*w}} \tag{2.4}$$

To shift the activation function to the left or right, we add bias to the activation function [2]. The new function will be 2.4, and as we can see from the e's exponent, bias does not affect the activation function if $b = 0$. As shown in Figure 2.5, if bias is a negative

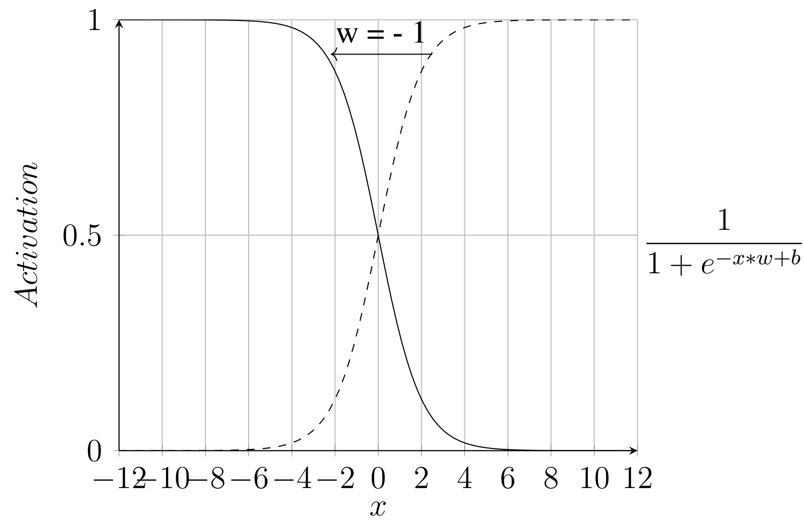


Figure 2.4: Sigmoid function with weight changing

value, the whole graph is shifted to left.

$$S(x) = \frac{1}{1+e^{-x*w+b}} \tag{2.5}$$

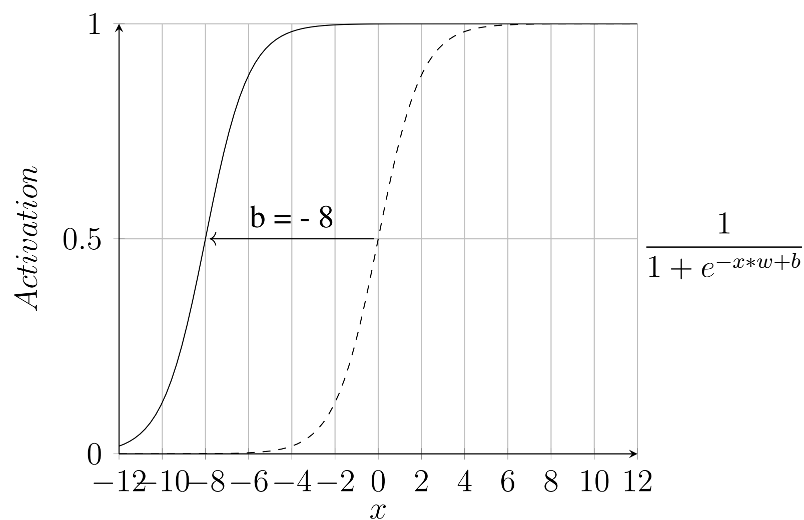


Figure 2.5: Sigmoid function with bias changing

One commonly used activation function is called Rectified Linear Unit (ReLU) [6]. ReLU is not computationally expensive, because as we can see in equation 2.6, given input will be returned if it is larger than zero, otherwise the output will be zero. ReLU's

ability to return zero value allows neuron to be in inactivate state, unlike with the sigmoid activation function. [6]

$$f(x) = max(0, x) \tag{2.6}$$

2.4 Neural network layers

Neural networks are collections of different types of layers. These layers are commonly divided into three (3) distinctive types: *input*, *hidden* and *output* layers. First of the three, input layer, is the starting point for the neural network. This layer takes the raw input and feeds it to the subsequent layer in the neural network which is usually a hidden layer. Second type of layers in the neural network is the output layer. Output layer's job is to accept the input from the earlier layer and represent the network's result for the original input. Between input and output layers are hidden layers. Hidden layers are the ones doing the mathematical calculations for the input. [7]

Neural networks' ability to learn complicated solutions is thanks to hidden layers. The output layer is also capable for learning just like hidden layers. Hidden layers can learn to find how to for example split data points into different categories when linear solution is not enough. In figure 2.3 we see a scenario, where hidden layers could assist us to divide data points into classes. Hidden layers can additionally be used for example to extract different features from an image. Convolutional layer is hidden layer type that is used to extract different . How convolutional layers work will be discussed in chapter 3. [7]

To move values or features from layer to layer, different connection types can be used. The simplest connection type is called *Dense layer* (Also known as Fully-connected layer) [8]. Layer is called dense layer if every neuron has a connection to every neuron in the previous layer [8]. So, if layer L_n has five (5) neurons and the next layer L_{n+1} has four neurons and is dense layer, then there are total of $5 * 4$ connections into layer L_{n+1}.

Another commonly used connection type is called as *Residual connection* developed by He et al. in 2016 [9]. Residual connections are sometimes referred as *skip-connections*. This type of connection allows neural networks to take a copy of layer's output and pass it to a layer which is deeper in the network. Passing these values allows them to skip a part of the network. [9]

Values in the network can become quite large. Large values in the neural network can cause problem when training it. One way to prevent these large values is to add normalization to the network. Normalization usually scales values to be between zero and one. In 2015 Ioffe and Szegedy introduces *Batch Normalization* to normalize layers output [10]. Batch normalization takes activations of a layer and applies a normalization for these activations. From equation 2.7 we can see how the normalization is done. In the equation x indicates the activation given as an input. μ is the mean value of the batch and σ is a standard deviation. Standard deviation σ is calculated by first calculating the mean value from the given values. The mean value is then subtracted from each value. After subtraction, results are squared. Standard deviation is now calculated as a mean of these squared values. Values γ and β are arbitrary values and are used to scale and swift the input. Values used for γ and β are learned during the training. [10]

$$(\frac{x - \mu}{\sigma} * \gamma) + \beta \tag{2.7}$$

Chapter 3

Convolutional neural network

3.1 Introduction

One of the goals of the computer science has been to teach computer to see. Many different methods have been researched, but in 1989 LeCun et al. [11] introduced new type of neural network that is able to extract features from a given image. This network utilize convolutions to find unique features from the image. LeCun et al. used this network to solve basic character recognition, which in this case was to classify handwritten digits. This network is called Convolutional neural network (CNN) which is a type of neural network. [11]

Using convolutions in a neural network did not became popular approach in over 20 years after the original release. In 2012 Krizhecsky et al. [12] released a paper about a model using deep learning with convolution. This model showed great success with ImageNet data set [12]. After this paper, the amount of research papers regarding the *Convolutional neural network* has increased, as we can see from Figure 3.1. CNN is capable for learning spatial hierarchies of different features from grid-like data, for example images. Key element of CNN is called a *convolution layer*, which is a combination of mathematical operations. In this chapter we will go through the basic building blocks for a convolutional layer and how they work. In chapter 3.2 we go through how the convolu-

tion work and after that in chapter 3.3 we discuss how the convolutional network modify outputs. [7]

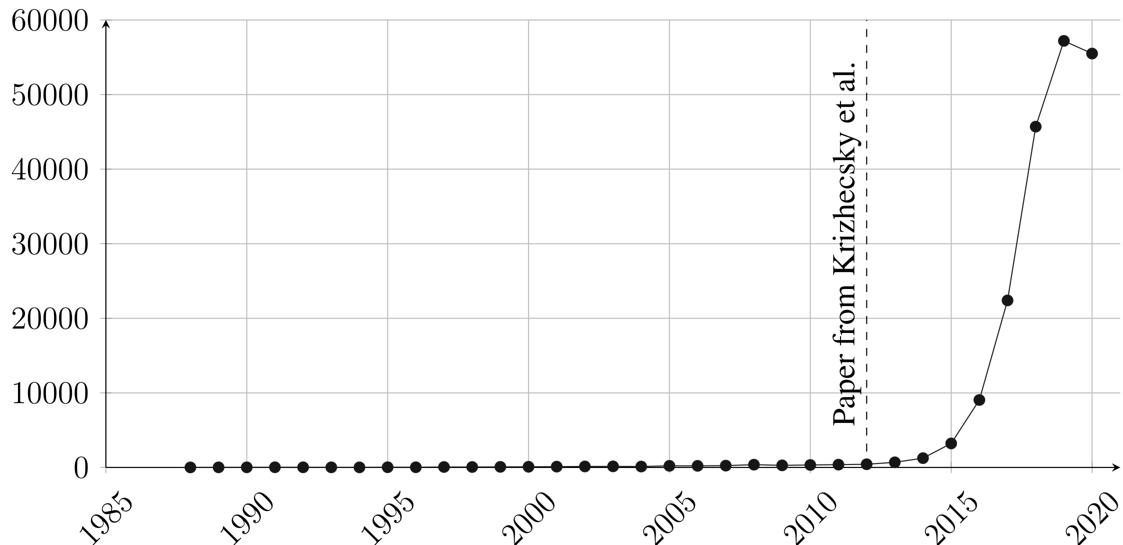


Figure 3.1: Amount of publication according to the search word *Convolutional neural network* in Google Scholar. Patents and citations were excluded from the search.

3.2 Convolution

Before we start to inspect convolutional neural networks, we must understand the main building block in this neural network. Convolutional neural networks contain layers which do convolutional operations. These layers are called *convolutional layers*. Operations done in convolutional layers' neurons are called convolutional between image and a *kernel*.

Kernel a small matrix which can be used to manipulate the target image. These manipulations can be for example, image sharpening, blurring and edge detection. The kernel will be slid across the image matrix, where multiplication operations will be done. Convolution can be used for feature extraction. For example, we can extract edges from the given image by using *Sobel operator* created by Sobel and Feldman in 1968 [13]. So-

$$G_x = \begin{bmatrix} +1 & 0 & -1 \\ +2 & 0 & -2 \\ +1 & 0 & -1 \end{bmatrix}, G_y = \begin{bmatrix} +1 & +2 & +1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix} \tag{3.1}$$

Figure 3.2: Sobel kernels for X and Y [13]

bel operator has two 3x3 matrices, one for vertical edges (G_x in Figure 3.2) and one for horizontal (G_y in Figure 3.2) edges.

$$G = \sqrt{{G_x}^2 + {G_y}^2} \tag{3.2}$$

In convolutional neural networks, convolutions are used with a *sliding window algorithm*. The kernel will be moved from left to right by the step size defined as a stride. Matrix multiplication is done, and the product matrix is summed up. The value of a convolution will be stored into output's matrix in corresponding space. Kernels' values used in Convolutional neural network are learned during the network's training.

$$O_{width} = I_{width} - G_{width} + 1 \tag{3.3}$$

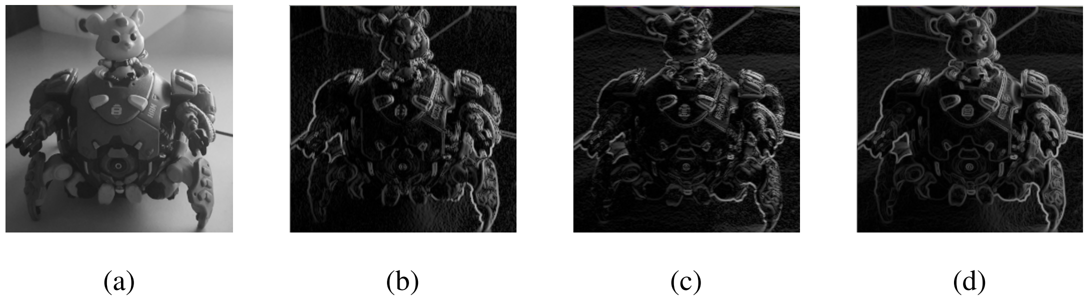
$$O_{height} = I_{height} - G_{height} + 1 \tag{3.4}$$

By default, the output matrix's dimensions are smaller than the original after the convolution. This is because there will be sections which are smaller than the given kernel. In equations 3.3 and 3.4 we can see how output matrix's dimensions can be calculated.

If we want to preserve the original dimensions, we must add padding to the image. In Figure 3.3, we can see how zero padding can be added to the input matrix to preserve original dimensions.

$$M = \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix} => \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.3: Adding padding to the matrix M



(a) (b) (c) (d)

Figure 3.4: Sobel edge detection. Picture *a* is an original grey-scale image. In picture *b* we can see vertical edges extracted with Sobel kernel G_x from Figure 3.2. In picture *c* we can see horizontal edges extracted with Sobel kernel G_y from Figure 3.2. In the last picture (*d*) we can see extracted edges after combining Sobel kernels with equation 3.2.

3.3 Pooling

Typically, after convolutions and non-linear activation comes a *pooling layer*. This layer's task is usually to summarize the given inputs, usually previous layers activations. Pooling helps us to add some position invariant, so if the input changes a little, it should not affect the pooling layers output. [7] Pooling layer will almost always downscale the input because of its nature. This is caused by the attempt to summarize the given inputs. Just like in convolutional layer explained earlier (chapter 3.2), many of the pooling methods are using sliding window algorithm and filters. [7]

One of the most used pooling methods is called *max pooling*. Max pooling selects the

largest value in a given space. From figure 3.5 we can see an example about how max pooling works and how it down scales the input. One of the strengths of max pooling is that it is not sensible to values exact location, assumed that the value is in the pooling region. [7]

$$\begin{bmatrix} \boxed{\begin{matrix} 4 & 2 \\ 0 & 7 \end{matrix}} & \begin{matrix} 3 & 9 \\ 3 & 6 \end{matrix} \\ \begin{matrix} 1 & 9 \\ 7 & 0 \end{matrix} & \begin{matrix} 4 & 1 \\ 8 & 2 \end{matrix} \end{bmatrix} \longrightarrow \begin{bmatrix} \boxed{7} & 9 \\ 9 & 8 \end{bmatrix}$$

Figure 3.5: Max pooling with a kernel of $2x2$ and stride of 2.

Chapter 4

Object Recognition

4.1 Introduction

Object detection is a challenging task in computer vision. The main goal of object detection is to detect instances of different objects from the given digital image. These detected objects are from predefined classes such as different vehicles. The purpose of object detection is to provide information on what the given image has and where on the image objects are, to different computer applications. [14]

When inspecting object detection from the perspective of computer applications, object detection can be divided into two (2) different topics: general object detection and detection applications. General object detection tries to create an unified framework, which is capable to simulate the human vision. Detection applications are used for specified areas, for example to detect vehicles from images. Object detection is used in multiple real-world applications thanks to the breakthrough in deep learning. [14][15]

4.2 Why is the object recognition important?

Object recognition has been integrated into our everyday life. Almost everything related to images and videos contains some form of object recognition. Usually, it is used to

detect different features from the images.

For example, self-driving cars are using object recognition [16]. Self-driving cars are using object recognition to figure out the surrounding environment. Detecting different vehicles and pedestrians is critical for having a working self-driving car. Additionally, object recognition is used to detect driving lines and traffic signs to help the car to navigate in the road. Of course, self-driving cars have multiple different sensors to work with object recognition systems.

Object recognition can be used to analyse images and videos. These analyses can be detections of objects of interests, for example vehicles or pedestrians. Detections provided by object recognition can be post-processed to get desired information. This information can be something like people count in a shopping mall.

4.3 Image classification

A task where the target is to recognise what is in the given image input is called image classification. Tries to answer following question: *"What is this picture of?"* This task is often the underlying problem in many different object recognition tasks. Image classification is, for example, a subtask in object detection (Chapter 4.4).

Image classification is widely used and it is seamlessly integrated into our lives. For example, face recognition in our phones and computers use image classification to figure out if we are in the picture. Images in large image galleries are categorised automatically into different collections. Our phones' camera applications use image classification to determine what is in the captured view to select correct filter to enhance our food pictures.

4.4 Object detection

Object detection is a task which answers to a following question: "What objects are where?" [14]. This task is a critical one in multiple different implementations used nowa-

Figure 4.1: Face detection done with Haar Cascades

days. Object detection combines the image classification we discussed earlier in chapter 4.3 with *object localization*. Object detectors usually returns us a *bounding box* and a class for found objects. Bounding box is a rectangle, which tries to border the detected object as well as possible. In this work, object detection will be covered more extensively in later chapters (6) because this object recognition type is one of this works main focuses.

Object detection is used everywhere just like image classification we discussed in previous chapter. This task can be used as a pre-processing for some computationally expensive operations. For example, for facial recognition systems, we could use light and fast object detection an algorithm to extract faces from an image. Then these extracted faces would be fed to algorithm, which handles the recognition task. In earlier chapter 4.3, I mentioned image classification in our mobile phones to determine what filter to use. Some applications draw a rectangle around faces in the picture. It is very likely that behind the scenes, the application use *Viola–Jones object detection framework* or something similar to quickly find every face [17]. This algorithm specializes for detecting faces from an image, so it is an excellent example about real world application for object detection. Because this work does not focus on face detection, we will not go through Viola–Jones object detection framework. To find more information about this algorithm, I recommend reading the original paper *Rapid object detection using a boosted cascade of simple features* by *Paul Viola and Michael Jones*.

4.5 Semantic Segmentation

Figure 4.2: Semantic segmentation done to a picture. Every pixel is labelled as a person or as a background. [18]

Semantic segmentation is an *image segmentation* task. The goal is to label each and every pixel in the input image. Unlike object detection, semantic segmentation is unable to differentiate instances of a same class. For example, if we do semantic segmentation for an image of a forest, we are unable to tell how many trees are in the image. If we are interested in these instances, then we are talking about the other task in image segmentation called *instance segmentation*.

To continue our theme regarding the real-world application examples, semantic segmentation can be used to modify an image. For example, semantic segmentation can be used to find background from the image and use this information to blur the background.

4.6 Instance segmentation

Instance segmentation is almost like semantic segmentation (chapter 4.5) which is combined with object detection (chapter 4.4) When we are doing instance segmentation for a given image, we do not try to label every pixel in the image. Only the pixels in the image

Figure 4.3: Instance segmentation where only pedestrians are detected. [18]

belonging to classes which we are interested in. Instance segmentation also differentiates different objects with the same labels. This allows us to find different trees from the forest picture.

There exists a third task for instance segmentation called *Panoptic Segmentation* which tries to label every pixel in the given image just like the semantic segmentation, but also differentiates different instances in the same class. So panoptic segmentation is a combination of semantic segmentation and instance segmentation [19].

4.7 Cleaning detections

Some of the object detection methods provide bounding boxes that cover the target object multiple times. We do not want to detect the same object from the given image multiple times. Instead, we want to get the best possible bounding box for the detected object in the image. Another thing we want to get rid of is detections with very low confidence score. In another word, post-processing needs to be done to the detections to get desired results from the detector.

One of the commonly used post-procession algorithm for cleaning up detections is

Non-Maximum Suppression (NMS) [20]. NMS is a simple algorithm, which requires following information about the detections: information of a boundary box and the confidence score about the predicted class. Both of these values are provided as object detector's output. Confidence score indicates how sure is object detection model that in the given bounding box exists an object. Additionally, there are two hyper-parameters which can be used to fine-tune the NMS. First one is the threshold value for the minimum confidence score T_c and the second one is the threshold for minimum Intersection over union T_{iou}.

When bounding boxes are fed to the NMS, it first removes the detections with a confidence score lower than the defined threshold T_c. After the initial cleaning, NMS selects the bounding box with the highest confidence score. NMS compares every other bounding box to the selected one and calculates the IoU. If the IoU is larger than the defined threshold T_{iou}, the compared bounding box is removed. The bounding box is defined to contain a separate detection from the selected one. When every bounding box has been checked, NMS selects a new bounding box from the remaining ones as earlier. This cycle is repeated until no bounding boxes remains. If there are multiple confidence scores for multiple different classes, the defined cycle is done for every class.

(a) Original ground-truth detections (b) Detections after applying NMS

Figure 4.4: With T_{iou} of 0.5, 10 true positive detections were lost after NMS

Rothe et al. calls this kind of NMS as *Greedy Non-Maximum Suppression* [20]. According to their paper, greedy NMS has three major problems. First problem they describe

is, that the top scoring bounding box may not be the best fit for the object. As described earlier, NMS selects the bounding box with the highest confidence score. It is possible that the bounding box is not fitting the object as well as other candidates with lower confidence score. Second problem is NMS may suppress nearby object with the same class. This is because the NMS removes other detections overlapping the selected bounding box. In figure 4.4a, we can see 51 ground-truth detections, but after the NMS we only have 41 detections left in figure 4.4b when the T_{iou} is 0.5. Third and the last problem is that NMS does not suppress false positive detections. Rothe et al. propose an alternative approach for the NMS to prevent those problems [20], but that implementation is out of the scope of this work.

4.8 Evaluation of object detectors

To compare different object detectors, we need a have a common way to do the comparison. Object detection covers two distinct goals, so evaluation is nontrivial. We must evaluate how well does the object detector classify objects from the given image and how well it can localize those objects.

Before we start discussing about different evaluation methods, we must familiarize ourselves about a few basic things. True positive (TP) shows us the number of detections done by object detector with a match in the ground-truth. False positive (FP) indicates the number of missed detections in the ground-truth and the false negative (FN) tells the number of detected objects not in the ground-truth. *Precision* and *recall* are two different measurements which indicates different evaluation target. Precision (equation 4.1) measures the object detectors ability to detect objects correctly from the image [21]. Recall (equation 4.2) evaluates how well does the object detector detect every object from the image [21]. For different use-cases we may want to focus one of these two metrics. For example, object detector used for self-driving car should have high recall to detect

any pedestrian from the image.

$$Precision = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall = \frac{TP}{TP + FN} \tag{4.2}$$

Intersection over union (IoU) is an evaluation metric, which is used to evaluate object localization done by object detector. To calculate IoU, we need to have the ground-truth for the bounding box and the predicted bounding box for an object. IoU can be calculated by dividing the union of these two bounding boxes by their intersection. In equation 4.3 we can see how IoU can be calculated, when bounding box for ground-truth is B_t and B_p is for predicted bounding box. Usually, prediction done by object detector is labelled as correct if the IoU is larger or equal to 0.5. This threshold can be changed and the higher it is, the more accurate the predictions need to be. [22]

$$IoU = \frac{B_t \cap B_p}{B_t \cup B_p} \tag{4.3}$$

Mean Average Precision (mAP) is a measurement that tells us how well our model performs for object detection. mAP is very commonly used and for many commonly data sets, many models have their mAPs already calculated. This allows us to easily compare our models with others. mAP should not be confused with *precision* (Equation 4.1), even if the name mean Average Precision suggests otherwise. To calculate mean Average Precision, we first need to calculate *precision* and *recall*, with slight modifications for TP, FP. Prediction is True Positive if the predicted bounding box and the ground-truth bounding box has Intersection over union over a threshold t. Otherwise, the prediction is classified as a False Positive. Threshold value for the IoU t is usually 0.5. For calculating the mAP, we need to calculate average precision for each class in the data set. Average precision is calculated by going through each detection for a class and calculating the precision and recall values as shown in equation 4.4. After that we calculate the average of precision for true positive detections. In equation 4.4 n indicates the number of true positives and

$Precision(i)$ is the precision for nth true positive. Finally, when we have calculated the average precision for every class, we take the mean of the average precisions to get the mean Average Precision score. In equation 4.5 we can see how the mAP is calculated. N is the number of classes and $AP(i)$ is the average precision for the class. [23]

$$AP = \frac{1}{n} \sum_{i=1}^{n} Precision(i) \tag{4.4}$$

$$mAP = \frac{1}{n} \sum_{i=1}^{n} AP(i) \tag{4.5}$$

Chapter 5

The Multiple Object Tracking Benchmark

5.1 Introduction

When evaluating and comparing object detection algorithms, we can use large data sets of images with given ground-truths. This makes the task of comparing different methods quite trivial. Unlike object detection, evaluation and comparing of different Multiple Object Tracking (MOT) methods is non-trivial [24]. According to Leal-Taixe et al., this task is non-trivial because of many reasons. For MOT, the first one being that it is hard to define distinctly what represent the perfect solution. Second one is that there were no widely used evaluation metrics to give quantitative results for comparing different methods across different publications. The third problem is that there were no pre-defined tests and training data to be used for comparing different methods. [24]

In 2014 Leal-Taixe et al. introduced *MOTChallenge* benchmark [1] to solve the problems mentioned earlier. MOTChallenge provides publicly available data set with total of 22 sequences containing 11286 frames. Data set has precomputed detections and annotations for each of the data set's frame. After releasing the initial data set in 2015,

team behind MOTChallenge has released many new data sets [25][18] to be used in MOT evaluations.

5.2 MOTChallenge data format

MOTChallenge provides testing and training data with ground-truth annotations. In MOT-Challenge 2015 data set, every image in the data set is in JPEG format and are named after 6-digit file name format sequentially (e.g., *000001.jpg*, *000002.jpg*). Detections and annotations are provided as CSV files. Every line in these CSV files are representing one instance of an object and are given as ten (10) values. [24]

The first value in a row represents the frame where the instance appears. [24] Usually, entries in the CSV files are sorted by this first value. This means that the first frames detections are at the start of the file and vice versa for the last frame of a sequence. Second value is the unique ID assigned to a trajectory. This ID is set to -1 in detections CSV file for every instance. Values three (3) to six (6) are indicating instance's bounding box. Values in positions three and four are indicating the bounding box's top-left corner in the 2D space and the values in positions five and six are telling the bounding box's width and height. The seventh value indicates the confidence score of the detection. Unlike in the detections CSV, in ground-truth and results file, the seventh value is used as a flag. If this flag's value is zero (0), it means that the instance from this row is ignored in the evaluation. On contrary, if the value is one (1), it means that the instance is active, and it will be used in the evaluation. The last three values in the row are representing instance's legs' location in the 3D space. These values are set to -1 for 2D tracking and are meant to be ignored. [24]

14, 9, 451, 171, 31.03, 75.17, 1, -5.6701, -6.8765, 0

14, 15, 337, 201, 30.467, 85.209, 1, -9.4426, -6.2248, 0

14, 19, 511, 237, 36.491, 84.883, 1, -9.9105, -10.482, 0

Figure 5.1: Visualization of values for PETS09-S2L1 frame 14

The data format was changed in *MOTChallenge 16* and *MOTChallenge 17* data sets. File format and the naming convention did not change, but the values in rows were changed. Instead of ten values like in *MOTChallenge 15*, the number of values was reduced to nine (9) values. The order of values did not changing, but two last values meaning were changed, because rows do not contain anymore information about the 3D space. The eight (8) value indicates the object's type. Type is given as an integer, and it is following the conventional given in the data set. Ninth (9) value shows how much of the bounding box is visible. This value can be lower than 1 if e.g., another object covers the instance. For detections file, two last values are ignored, so they are set to -1. [1]

When inspecting MOT20 dataset, there were problems with the data. Data set's provided detections did not match the description given in its official paper *MOT20: A benchmark for multi object tracking in crowded scenes*[18]. Detection files should contain according to the paper nine (9) values per line which is the same protocol as used in MOT16 data set. Detections provided in MOT20 data set are following the same format as used in MOT15, so one line contains ten (10) values. Following detection lines are from MOT20 data set's public detections.

$$1, -1, 950, 520, 58, 142, 0, -1, -1, -1$$

$$2, -1, 667, 684, 99, 223, 1, -1, -1, -1$$

As we can see from those two lines, data set does not follow the protocol proclaimed in *MOT20: A benchmark for multi object tracking in crowded scenes*[18]. This means that detections do not have a confidence score.

5.3 Evaluation

MOTChallenge's purpose is to be a fair platform for comparing different tracking methods. This is achieved by providing equal conditions for each method starting from ground-truth data to evaluation metrics. [1] Laura Leal-Taixe et al. chose three sets of measurements for comparing different methods tracking performance to give more ways to compare different methods. For measuring frame-to-frame performance, team selected CLEAR-MOT proposed by Stiefelhagen et al. in 2006. Second selected measurement proposed by Wu and Nevatia in 2006 focuses on tracking quality of the tracking method. The third and the last measurement is IDF1 proposed by Ristani et al. in 2016 is trajectory-based measurement. Combination of these three methods gives a good view on different tracking methods' performance. [1]

Evaluation of frame-to-frame matching between predictions and ground-truth is done with Multi-Object Tracking Accuracy (MOTA) which represents the CLEAR-MOT. MOTA is used to evaluate the tracking performance locally, so it penalizes for example identity switches between frames. MOTA tells us a summarized version of three sources of error as we can see in the equation 5.1 [1].

$$MOTA = 1 - \frac{\sum_t (FN_t + FP_t + IDSW_t)}{\sum_t GT_t} \tag{5.1}$$

In the equation 5.1, FN is the number of false negatives, in another words, the number of objects not detected from the ground-truth detections. FP is the number of false positives, which tells us the opposite of FN: The number of detections not included in the ground-truth. $IDSW$ tells the number of identity switches and the GT is the number of ground-truth detections. Frame index is indicated as a t, so a following marking GT_t can be read

as *the number of ground-truth detections in frame* t. MOTA score for a tracker is reported as a percentage between $(-\infty, 100]$. [1]

To measure tracking method's precision, team selected Multiple Object Tracking Precision (MOTP). MOTP tells us about the average dissimilarity between true positive detections and ground-truth targets. So MOTP measures localisation precision of a tracker method. To calculate the MOTP for detections with bounding boxes, equation 5.2 is used.

$$MOTP = \frac{\sum_{t,i} d_{t,i}}{\sum_t c_t} \tag{5.2}$$

In equation 5.2 [26], $d_{t,i}$ tells how well the predicted bounding boxes overlaps the object i and its ground-truth. c_t denotes the number of matches. So, as we can see from the equation 5.2, MOTP measures the average overlap of detections and corresponding ground-truths. Because MOTP quantifies the localization precision, it does not tell much about the tracker if it is Tracking-by-Detection (TBD) tracker. Instead MOTP tells us how does the detector work.

To contrast CLEAR-MOT, team selected Identification F1 Score (IDF1) act as an identity-based measure. IDF1 measures how well does the tracking method perverse identities over the entire sequence. To calculate IDF1, we must first solve bipartite matching problem to map predictions to ground-truths by connecting pairs with the largest temporal overlap [1]. Solving bipartite matching problem means that we have two groups, and we try to find pairs for values from the groups. To goal is to find pairs where each value only has one pair from another group. Values in the same group cannot be pairs. After solving the problem, we can compute the number of False Positive ID (IDFP) [27], False Negative ID (IDFN) [27] and True Positive ID (IDTP) [27]. With those values, we can calculate the IDF1 with equation 5.3. [27]

$$IDF1 = \frac{2 * IDTP}{2 * IDTP + IDFP + IDFN} \tag{5.3}$$

Team also added two more measures that utilizes values calculated earlier which are proposed by Ristani et al. [27]. Identification Precision (IDP) measures the faction of

computed ground-truth detections which are correctly identified [27] by the tracker. IDP can be calculated with following equation (5.4) [27].

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{5.4}$$

Identification Recall (IDR) can be calculated with equation 5.5 [27].

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{5.5}$$

The final type of measures focuses on tracking quality. These values are qualitative and evaluate the percentage of ground-truth trajectories found by tracking method. For each ground-truth trajectory, we can calculate one of the following labels: Mostly tracked (MT), Partially tracked (PT) and Mostly lost (ML). According to Wu and Nevatia [28], a trajectory is labelled as MT if and only if 80% of its life span is tracked successfully. The range between $(20\%, 80\%)$ will give a trajectory a label of PT[28]. If the success rate is 20% or less, the trajectory is labelled as ML[28]. Success rate indicates how well does the tracking algorithm keep the tracked trajectory to be same as the corresponding trajectory in the ground-truth [28]. MOTChallenge reports MT and ML as a ratio to a total number of trajectories in the ground-truth. [1] The desired result is that the number of MT is high and the number of ML is low. It is also important to remember that this measurement does not care if the object ID does not remain the same. [1]

MOTChallenge provides a metric how well does the tracking algorithm track long sequences without any gaps. This measurement is called track fragmentation (FM) and it tells how many times the ground-truth trajectory was untracked by the tracker. [25]

Chapter 6

Object Detector

6.1 Introduction

When trying to track objects from a video, usually paradigm called Tracking-by-Detection (TBD) is used. TBD can be used to inspect given video in two (2) different scopes: *microscopic* and *macroscopic*. According to Leal-Taixé microscopic scope focuses in detection of individuals. Individual's features and motions are under inspection and overall motion found in the image is not inspected. Macroscopic tracking tries to capture the flow in the given image and focus is more global, not like in the microscopic tracking. An example scenario for macroscopic tracker would be a detection of movement's flow of the crowd. In this work, we will be focusing on microscopic trackers. [29]

Tracking-by-Detection is created from two (2) different parts: Object detector and Data Association. We will first discuss about object detectors and how they fulfill their tasks. Object detectors are the first module in TBD. This module's job is to extract different objects bounding boxes and class probabilities from given input image. Class probabilities are a list of probabilities indicating how sure is object detector that the object is belonging to the class. For example, if our object detector has been trained to detect three different classes, then the output would be three values. [29]

The second module in Tracking-by-Detection takes the detections and their informa-

tion as an input. The second module is tracking algorithm, which job is to create associations for detections between frames. Tracking algorithms create unique identifiers and tries to track objects between frames. This task of tracking where object with an id is in the next frame is called data association. We will discuss more about different tracking algorithms and data associations later in chapter 7. [29]

Object detector is one of the two key components of Tracking-by-Detection paradigm. Its job is to localize and identify objects from the given input image. Object detectors can be implemented by using classical machine learning or with a Deep neural network. Histogram of Oriented Gradients (HOG) is an object detector which can be viewed as object detector which is using classical machine learning, when HOG is paired with Support Vector Machine (SVM). HOG and SVM will be covered in later chapter 6.2.

Object detector which is implemented by using Deep neural network is covered in this work, two object detectors will be covered. First one is Region Based Convolutional Neural Network (R-CNN) which can be considered as a more classical deep neural network model. The main idea of R-CNN will be discussed in chapter 6.3 and its advanced version *Faster R-CNN* will be also described in that chapter. The more modern model is called You Only Look Once, which is faster than R-CNN. YOLO and its advanced version YOLO V3 will be inspected in chapter 6.4. Differences between these to DNN methods will be discussed at the end of chapter 6.4.

Object detectors are usually split into two groups: *one-stage* and *two-stage* detectors [30]. Two-stage detectors use different modules to generate detections from the input. In the first stage, there is usually a module that outputs different region of interest, which are then given to the second stage. Second stage handles the object classification and fine-tuning of the bounding boxes, in another words, the second stage is trying to classify what is in the region of interest. One of the commonly used two-stage detectors is Faster R-CNN, which we will discuss in chapter 6.3. Two-stage detectors are capable for reaching high accuracy but are commonly slower than one-stage detectors. Compared to two-

stage detectors, one-stage detectors usually have lower accuracy, but are faster. One-stage detectors are solving regression problem instead of the classification problem. They do not take any region of interest proposals. You Only Look Once (YOLO) is one commonly used example for single-stage detectors. [30]

6.2 Histogram of Oriented Gradients and Support Vector Machine

Histogram of Oriented Gradients (HOG) is a *feature descriptor*, which can be used to extract gradients and orientations from an input image. Features extracted with HOG was first introduced by Dalal and Triggs in 2005 [31]. HOG outputs multiple histograms which are generated from multiple small regions from the original image. Histograms are created from calculated gradients and orientations from the specified region.

Histogram of Oriented Gradients does not really care about the input image's size, but the original paper [31] use an aspect ratio of $1 : 2$ and the image dimensions were 64x128. These dimensions allow us to easily split the image in 8x8 regions. Gradients are calculated for x and y directions by using a convolution, which has been covered in chapter 3.2. For this convolution HOG use kernel of size 3x1: $[-1, 0, 1]$ and $[-1, 0, 1]^T$ to calculate the gradients for X and Y.

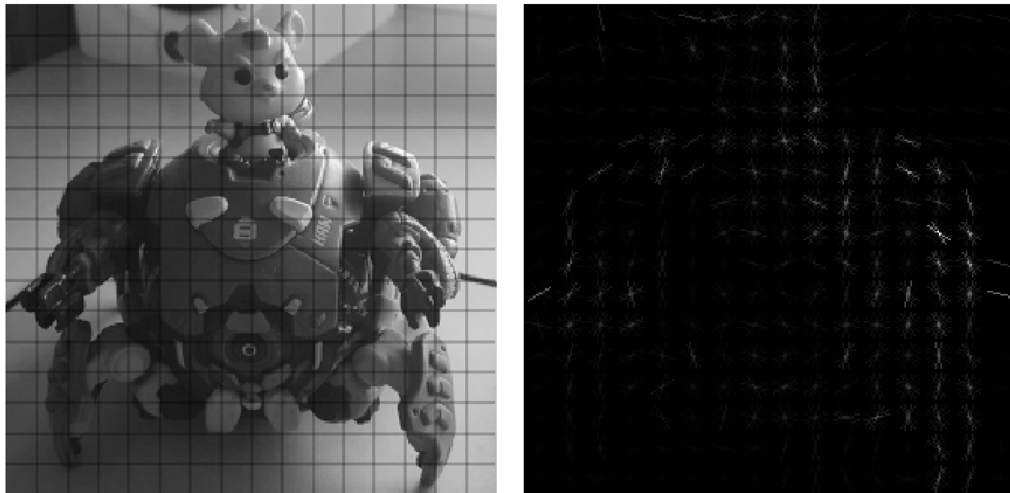
Calculating the magnitude for the gradient can be done with Pythagoras theorem (6.1).

$$v^2 = {G_y}^2 + {G_x}^2 => v = \sqrt{{G_y}^2 + {G_x}^2} \tag{6.1}$$

For each pixel we know the change in X and Y axis, so we can calculate the orientation of a gradient in degrees by using equation 6.2.

$$\theta = \arctan \frac{G_y}{G_x} \tag{6.2}$$

Now that we have our two (2) matrices for magnitudes and orientations, we form histograms of nine bins for each non-overlapping 8x8 area. Because we are using unsigned



(a) Image split into 16x16 regions (b) Histograms visualized

Figure 6.1: Visualization of how the histogram from regions could be visialized

orientations, our value range is 0 to 180. Our bins are split by orientation:

$$0, 20, 40, 60, 80, 100, 120, 140, 160$$

Values from the 8x8 region are moved to the created bins directed by gradient direction. The value itself is selected from gradient magnitude corresponding to the direction. If the direction is between two bins, the value is divided by the distance. So, for example if direction is 50 and the corresponding magnitude is 8, the value 8 is split evenly into bins 40 and 60. [31]

With histograms we have a collection of 8.1 vectors. These vectors contain some noise caused by lighting. Because of these variations, we do not have to normalize the vectors. For normalizing these vectors, HOG use L2 normalization also known as Euclidean normalization. With equation 6.3 we can calculate value which can be used as a divider to normalize every value in the vector. [31]

$$|x| = \sqrt{\sum_{k=1}^{n} |x_k|^2} \tag{6.3}$$

This normalization is not done for 8x8 regions, instead it is done for 16x16 regions containing four 8x8 regions. After normalization is done for the region, the region is moved

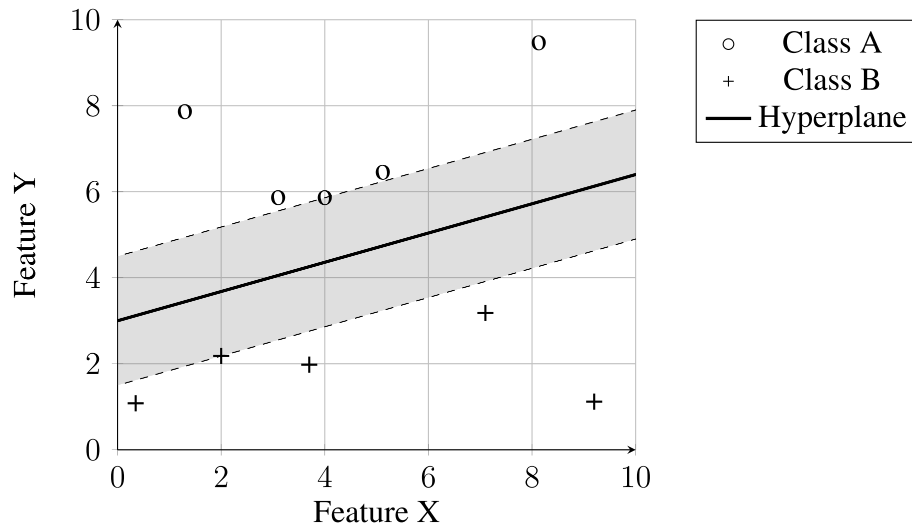


Figure 6.2: Visualization of SVM splitting data points

by 8 pixels and the process is repeated until every vector is normalized. After normalizations, the Histogram of Oriented Gradients feature vector can be created and fed to the Support Vector Machine. Each normalized $36x1$ vector is concatenated into one vector.

Support Vector Machine (SVM) (sometimes referred as Support Vector Network) is a supervised learning model used to do binary classification [32]. There also exists Multilabel Support Vector Machine. According to Noble [33], SVM consists of four basic concepts: hyperplane, maximum-margin hyperplane, soft margin and kernel function. Main idea of a SVM is to find a line that separates clusters of objects in a most optimal way. This splitting line is called a *hyperplane*, that same as a decision boundary discussed in chapter 2.3. Finding the most optimal hyperplane, we are trying to find the widest margin to split the data. To put this goal in another words, SVM tries to find a hyperplane which distance to data points is as far as possible. This gives as a *maximum-margin hyperplane*. [33]

In figure 6.2 we can see simple linear hyperplane dividing two class. Dotted lines above and below the hyperplane are in equal distance from the hyperplane. Coloured area in the figure represents the maximum-margin hyperplane and the data points on the line are called *support vectors*. [33]

Data is typically hard to split into distinct groups. Usually, data contains outliers, and these outliers will lead to problems with SVM. To handle this problem, we must allow few outliers. This is called *soft-margin* and with that, SVM tries to find the hyperplane while balancing between optimal split and minimising miss classifications. [33]

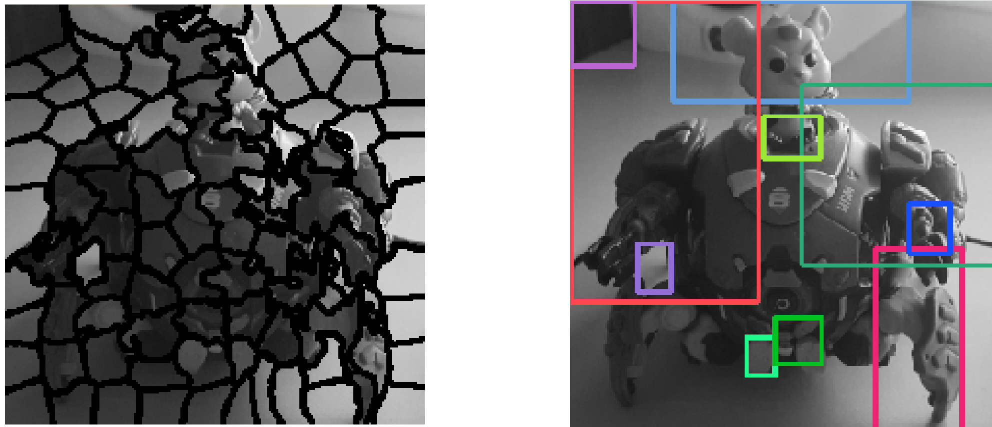
SVM is capable of using *kernel functions* [33] (sometimes referred as a *Kernel trick*) for creating new features and transforming the given data. Simplest kernel function is called a *linear kernel* function. This kernel function tries to find a linear line to separate the data like in the figure 6.2. As the name suggests, linear kernel function is not suitable for finding nonlinear solutions. For finding nonlinear solutions, SVM could use, for example, the popular kernel trick called *Radial basis function kernel (RBF)* [34]. RBF kernel enables SVM to inspect different data points relationship in higher dimension.

6.3 Region Based Convolutional Neural Network

Convolutional neural network (CNN) can be used just like Histogram of Oriented Gradients (HOG) for classifying an image. But when we discussed about object detection in chapter 4.4, we described two tasks for object detection: classification and localization. Old school way to achieve localization with CNN is to combine it with a sliding window algorithm. This approach is highly computationally expensive, because we would have to run every sliced part of the image through our network. Sliding window would also be very sensitive to our hyper-parameters. For 512x512 image and a kernel with a size of 32x32 we can calculate the number of passable images by using equations 3.3 and 3.4 (discussed in chapter 3.2). We get the total number of 231361 images when sliding window is moved with a step size of one each time.

In 2013 Uijlings et al. introduced optional method to replace sliding window from object detection solutions [35]. This method is called *selective search*. Proposed method suggests areas from the input image which could contain an object. The classification of

this area will be done with a classifier. In figure 6.3b, we can see few regions proposed by selected search. Selective search does not now what object should be found, instead it works with superpixels and creates regions. Superpixels can be imagined as a collection of pixels creating some sort of context, for example, pixels in superpixels might be similar in colour or another low-level feature [36]. In figure 6.3a, we can see how well the super-pixels' boundaries follows the character's outlines. Each super pixel is represented in the figure as an area with black outline. With the help of superpixels, selective search creates region proposals. The number of proposed regions is around 2000 proposals. Compared to the sliding window images, selected search return much fewer images.



(a) Visualization of superpixels

(b) Suggested regions with selected search

Figure 6.3: Visualization of different parts of selective search

Girshick et al. [37] proposed new object detector that utilizes CNN and selective search. This combination is in a paradigm called ”recognition using regions” [38]. This object detector is called Region Based Convolutional Neural Network and it can solve object detection and object segmentation (discussed in chapter 4.5) tasks. Girshick et al. implementation of R-CNN is simple as an idea. Select search extracts about 2000 region proposals from the input. These proposals are fed to CNN which outputs features to the classifier layer. Support Vector Machines are used to classify given features and to give the final classification to the given input. [38]

Region Based Convolutional Neural Network (R-CNN) is well performing, but it is not capable of performing in real time. This is because R-CNN is the convolutional neural network passing for each object proposal provided by region proposal algorithm . In 2015 Girshick et al. proposed a new version of R-CNN called Fast R-CNN. Fast R-CNN was created to reduce the R-CNN's object detection time, which is according to Girshick about 47 seconds per image [39]. The proposed network could do the object detection in 0.3 seconds, which is a significant difference for detection steps. [39]

The reason for the reduction of time used for detections is the architecture used in Fast R-CNN. Unlike the R-CNN, Fast R-CNN processes the whole input image to produce a new feature map. Generated feature map is used for each proposed region. For each proposed region, Fast R-CNN use Region of Interest (RoI) pooling to extract a feature vector from the region. Feature vector is then given as an input for fully connected layers that splits into two sibling networks. One of the sibling networks produces the estimations for different classes by using softmax, which differs from R-CNN that use Support Vector Machines. Other network gives us a refined bounding-box as four values. [39]

To reduce the total time used for extracting objects from the given images, bottleneck needed to be removed. Fast R-CNN fixed the time used for convolutions, but it did not touch the region proposal. Region proposal step is slow and is run on CPU, while the CNN part is run on GPU [40]. In 2016 Ren et al. [40] proposed a new R-CNN network, which solves the region proposal bottleneck. Network's name is Faster R-CNN, which utilize the Fast R-CNN, but replaces the region proposal step with a new neural network called Region Proposal Network (RPN) [40]. Paper's authors say that region proposal could be reimplemented for a GPU, but it would miss benefits of shared computing.

Region Proposal Network is almost like selective search when only inspecting input and output. RPN takes an input image and outputs regions as proposals where objects might be found, just like selective search or any other region proposal algorithm. Unlike selective search, RPN is a convolutional network. RPN use sliding window with a

small network which output is mapped to a lower dimensional feature vector. Features are then fed into two different fully connected layers to provide information for region proposals. These two layers are called Box-regression layer (REG) and Box-classification layer (CLS). REG provides the bounding boxes and CLS provides an *objectness* score, which measures how likely the suggested region is not background. Because RPN can propose multiple regions for each sliding window location, REG and CLS are capable also for providing multiple outputs. [40]

Region Proposal Network (RPN) use reference boxes called *anchors*. These anchor boxes are used as predefined bounding boxes with different sizes and aspect rations. Sliding window use every defined anchor box every time and are used to obtain first predictions for objects locations. The Box-regression layer will then refine the bounding box. In the paper authors used nine anchor boxes denoted as k. Because for each anchor box are processed by REG and CLS at the same time, the output sizes for these two layers are $4k$ and $2k$. [40]

For Faster R-CNN, the Region Proposal Network was fused with Fast R-CNN. The feature map generated by Fast R-CNN is used as RPN's input image. This fusion allows Faster R-CNN to use shared computing. RPN's output will be used for Fast R-CNN's RoI pooling. Thanks to the fusion of RPN and Fast R-CNN, Faster R-CNN does not suffer from bottleneck caused by region proposal step. RPN proposes 300 regions instead of around 2000 proposals usually outputted by selective search. Even with lower number of proposed regions, Faster R-CNN has high object detection accuracy. [40]

6.4 You Only Look Once

In 2016 Redmon et al. introduced a new approach for object detection [41]. You Only Look Once (YOLO) (also known as *YOLO V1*) is a deep neural network just like R-CNN discussed earlier in chapter 6.3, but is does not use regional proposal to find potential

bounding boxes. It is also good to know, that unlike Histogram of Oriented Gradients (chapter 6.2) and Region Based Convolutional Neural Network (chapter 6.3), YOLO does not use sliding window nor regional proposals. [41]

YOLO takes a new approach for the object detection, instead of trying to solve classification problem, it solves the regression problem. Unlike CNN-based networks, which look at the input image multiple times, YOLO only looks at the image once (as the name suggests). According to the authors, YOLOv1 is 2.5 times faster than *Faster R-CNN*. Reason for YOLO's speed is its simple structure. It contains only one convolutional neural network, which predicts multiple bounding boxes and class probabilities simultaneously. [41]

For predicting bounding boxes, YOLO uses whole image's features. Input image is split into NxN regions, and each region has their own responsibilities. Each region predicts a bounding box and a confidence score about how sure region is that in the given bounding box exists an object. Predicted bounding boxes are represented by four values: X, Y, W, H. X and Y are coordinates for the center of the predicted bounding box. Width and height are relative values from the whole image, so their values range from zero (0) to one (1). Additionally, each region predicts class probabilities for every known class. YOLO then uses Intersection over union to clean predicted bounding boxes. [41]

Half a year later Redmon and Farhadi introduced a first incremental upgrade for YOLOv1, called as *YOLOv2* [42]. According to the authors, YOLOv2 fixes YOLOv1's localization error as well its other shortcomings. Two of the many structural changes are addition of Batch normalization and a usage of Anchor Boxes. Authors of the paper were able to use YOLOv2 to create a network called *YOLO9000*, which was a state-of-the-art object predictor. YOLO9000 can detect over 9000 different classes in a real-time. [42]

Third version of YOLO was released in 2018 [43] bringing incremental changes to YOLOv2. Name of YOLO's third version is YOLOv3. YOLOv3 has better bounding box prediction and class prediction. For class prediction YOLOv3 replaced Softmax with

logistic classifier. Improvements for detecting object with different scales by doing predictions over three (3) different scales. This type of prediction is done by having upscaling in the feature extraction neural network. YOLOv3 was published in 2018 but is still one of the fastest object detectors with a reasonable accuracy. [43]

Chapter 7

Data Association

7.1 Introduction

Second part's job in Tracking-by-Detection is to connect different detections provided by object detectors between frames. To simplify this chapter, we will be using a term *tracker* for algorithms used for data association. Tracker's job is to assign identifiers for detections and track of them between frames. Good tracker can even track objects when object detection step fails for few frames, or the object is occluded by something. Tracker should be able to track multiple objects correctly, even when detectors fail to identify them in some frames.

Trackers' have challenges they need to handle. In my opinion, one of the most common problem is the *identification switch*, which usually occurs when objects overlap. Other challenges trackers face are occlusion of an object and imperfect detections. In this chapter we will go through a few different trackers. The order of trackers will be from simplest to most complex, according to my opinion.

Tracking algorithms are divided into two groups: *online* and *batch* trackers. Online trackers algorithms are capable for real time tracking. Detections can be fed to the tracker directly and it can associate those detections between frames. Because of this, online trackers can be used, for example, in autonomous driving applications [44]. Online tracker

algorithms, which are capable for working in real-time, are the main focus of this work.

If our goal is to process video sequence, we have access every frame in the sequence. Batch tracking algorithm can be used when we have access for every frame. As the name suggests, batch tracking processes the video sequences in batches, unlike the online trackers which are solving the data association frame-by-frame. Batch tracking algorithms are trying to usually solve tracking as a global optimization problem [44]. To learn more about batch tracking algorithms for MOT, I suggest reading *Multiple Object Tracking Using K-Shortest Paths Optimization* by Jerome Berclaz et al. [45]

7.2 Euclidean distance

In my opinion, the simplest tracker is an algorithm using *Euclidean distance*. This algorithm uses the Euclidean distance to connect object detections between each frame. To make the calculations easier, we are not using the bounding boxes for the calculations. Instead, we are using centroids for distance calculations.

Centroids are center points of provided bounding boxes. Calculating a center point is easy and fast for bounding box, because it is a rectangle. If bounding box is provided with coordinates for top left corner with width and height, we can just divide those dimensions by two (2). By using those remainders with given coordinates, we get the locations for centroid of that bounding box. To calculate the Euclidean distance between centroids, we use following equation for two dimensional space:

$$d(P,Q) = \sqrt{(P_x - Q_x)^2 + (P_y - Q_y)^2} \tag{7.1}$$

Where P and Q are coordinates in two-dimensional space.

This tracker assigns a new identifier for new objects and reassigns these identifiers to detections in a new frame by using the calculated distance. When pairing detections, we assume the nearest new detection is the same object that was detected in earlier frame. This causes a problem when detections overlap. Overlapping usually ends up with identi-

fication swapping. Because this tracker only cares about the distance, it does not know in what directions objects are moving. For examples if two pedestrians pass each other by moving opposing directions, Euclidean tracker will mostly swap their ids. This scenario assumes that both pedestrians are detected by the detector layer. In figure 7.1, we can see how a tracker using Euclidean distance would swap identifiers. When new detections are close to old detections, swapping can easily happen. This will also lead to problems when trying to track pedestrians in crowd. But usually, it is most likely that the person in front will be detected, but the one behind will not be detected. Euclidean distance tracker cannot handle missing detections by default, so described event would lead to identifier swapping for the person in front. But for the person behind, this will lead to completely new identifier.

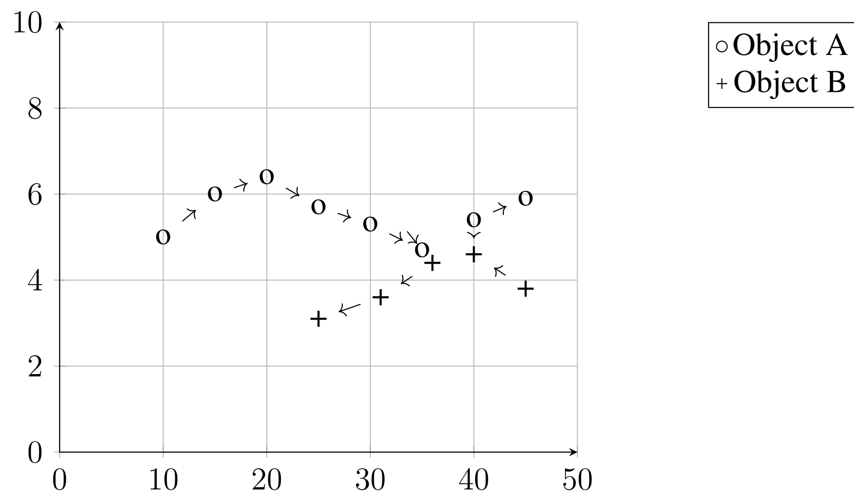


Figure 7.1: Visualization of identifiers switching for Euclidean distance

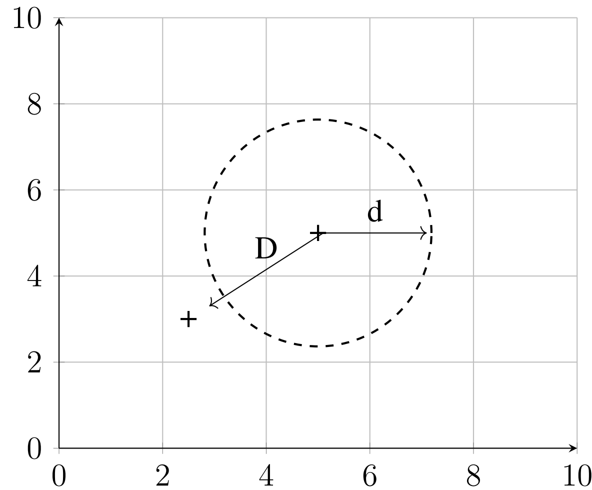


Figure 7.2: Euclidean distance tracking with visualization of maximum distance

Euclidean distance tracker does not know much about the object it is tracking. Because it only knows the last location, it cannot reidentify that object if we are missing a detection. So, what would happen if we lost the detection but in that same frame, we found a new object on the opposite side of the image? Tracker would assume that the new object is the same as the one we lost. We know that this should not be possible, so we must create limit for the maximum distance between detections. In figure 7.2, we have a visualization of this maximum distance. The variable d in the figure is a hyper-parameter which indicates the maximum allowed distance. It prevents tracker from connecting detections with arbitrary distances. In my opinion this is a solid improvement idea, but it has a noticeable drawback. The maximum distance is decided by us, so we can easily give it a bad value. Also because of this range, this tracker will struggle to track fast moving objects.

For single object tracking, we can just assume that the object in frame $T+1$ is same as in the frame T. As for Multiple Object Tracking we must reidentify objects in frame $T+1$ to match the ones in frame T. Let's assume our detections are perfect, and the amount of detection is equal between frames T and $T+1$. Simplest way to associate detections between frames with Euclidean distance would be to calculate distances between every

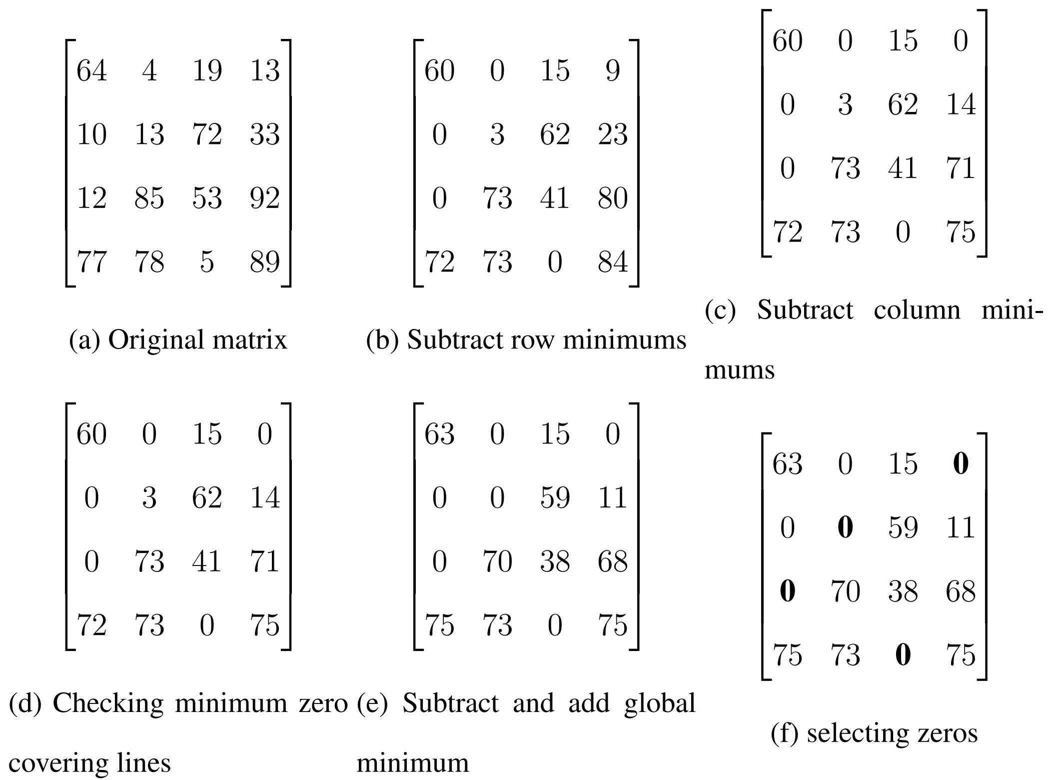
detection between frames. After these, calculations we would just associate detection by using shortest distance. To ensure our association is optimal, we should try to minimize the total sum of selected distances. This approach works, but it has a time complexity of factorial time ($\mathcal{O}(n!)$).

The task of associating old detections with new detections can be viewed as an assignment problem where we try to minimize the total distance between detections. Assignment problems can be easily solved with *Hungarian algorithm* developed by mathematical Kuhn in 1955 [46]. Hungarian algorithm tries to create pairs with minimal cost [47]. Let's call the matrix given to the Hungarian algorithm as a *Cost matrix*. To create the cost matrix, we need to have two groups. For example, first group is parts we need to fix a car and the second group is shops that sells those parts. The problem is that we can only buy one part per store. From the two groups we create a matrix a size of number of needed parts and the number of stores. Each row in the matrix represents a part from the parts groups and each column represents a store. We fill each row with a price of that part for corresponding store's column. After filling the whole matrix, we have a cost matrix which can be fed to the Hungarian algorithm to tell us where we should buy each part to spend as little money as possible.

Hungarian algorithm contains five (5) different steps to find the optimal assignments. Step 1 is to subtract minimums from each row. Step 2 is to subtract minimums from each column. Step 3 is to use a least number of lines to cover each zero. If the number of lines is less than the number of rows, go to step 4, otherwise go to step 5. Step 4 is to subtract the minimum value from the uncovered values and add it to the values which were crossed twice. Go to step 3. Step 5 is to select a zero value for each row and column. [47] Compared to the brute force method earlier, Hungarian algorithm's time complexity is only $\mathcal{O}(n^3)$.

Let's go the Hungarian algorithm through with an example case. We have four detections in frame $T-1$ and four detections in frame T. Let's create a 4 by 4 matrix and fill

it with the Euclidean distances, where rows represent the detections in frame $T - 1$ and columns are the detections in frame T. In Figure 7.3a, we can see the original matrix. Now we apply the first step to the matrix, and we subtract minimums from the rows. In this example, minimums are following for the rows: $[4, 10, 12, 5]$. After subtracting we get the matrix in Figure 7.3b. Next step is to subtract minimums from the columns. For columns, minimums are following $[0, 0, 0, 9]$. After subtracting we get the matrix in Figure 7.3c. After subtracting , we must find the minimum number of lines to cover each zero in the matrix. In this example, the minimum number of lines is three (3) as we can see from the figure 7.3d. Unfortunately, number of lines is less than the number of detections (4). Because of this, we must step 5. First, we need to find the minimum value which is not covered by any lines. In this case, the minimum value is three (3). We subtract the value from every non-covered field, and if the field is covered by two lines, we add the value to it. As a result, we get the matrix in Figure 7.3e. Now we repeat the step 3 and find the minimum number of lines requires to cover zero fields. The minimum number of lines is four, which is the same as the required amount. Finally, we move to the step 5 and select a column for each row as in the figure 7.3f. After selecting, we have the optimal assignments for detections.



(a) Original matrix (b) Subtract row minimums (c) Subtract column minimums

(d) Checking minimum zero covering lines (e) Subtract and add global minimum (f) selecting zeros

Figure 7.3: Hungarian algorithm steps

7.3 Kalman filter

Euclidean distance-based tracker has multiple different problems as discussed earlier in chapter 7.2. It cannot recover from imperfect detections and will always assign new ID for the detection. For handling this problem, we need to have an algorithm which can continue tracking even without detections. In another words, we need to find an algorithm which can predict where the detection should be in frame $t + 1$. One popular algorithm for solving this kind of problem is called *Kalman filter*.

Kalman filter was developed by Kalman in 1960 [48]. Kalman filter is an estimator, which has an ability to estimate system's error covariance. It also has an ability to improve system measurements recursively. Due to these features, Kalman filter has multiple different real-life implementations in several different field. One of the first real life

uses cases for Kalman filter was its usage to solve problem related to space navigation in Apollo Project in 1961 [49]. Additionally, Kalman filter is used in example global navigation satellite systems [50][51].

Kalman filter (also known as Linear quadratic estimation [52]) is a capable algorithm of providing estimations about different states. It uses previous measurements to create predictions how would the state change for the following measurement. Because Kalman filter's internal structure is quite a vast concept to cover, in this work we will only cover the key points. This allows us to gain a general understanding of how Kalman filter works and why it is still relevant estimator algorithm.

Kalman filter is capable of using noisy measurements to create accurate predictions from previous measurements. For creating state estimation for moment t, Kalman filter stores only information calculated from previous measurement $t-1$. After each measurement, the state is updated with the new measurement from moment t. To accomplish this, Kalman filter is working as a cycle between two different stages: prediction and update [53]. When in predict stage, Kalman filter outputs an estimation of the state before tuning the estimate with actual measurements in update stage. [48]

Kalman filter does not expect large changes for trackable objects motion. For example, as we can see from figure 7.4, Kalman filter's predictions are reasonably good for first few frames. When measurement's location changes unexpectedly, Kalman filter's prediction is far from the measurement. Kalman filter tries to correct its predictions and shown in the figure 7.4, predictions and measurements are close each other soon. When the movement of an object is not changing greatly, Kalman filter predicts quite accurately as can be seen from figure 7.5. Compared to predictions from figure 7.4, predictions are measurements, which are close to each other at all times.

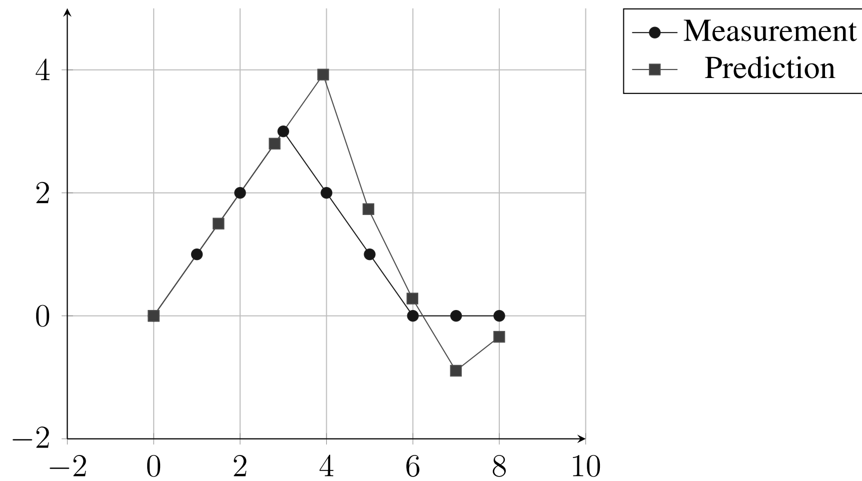


Figure 7.4: Kalman filter predicting objects location with constant speed from left to right

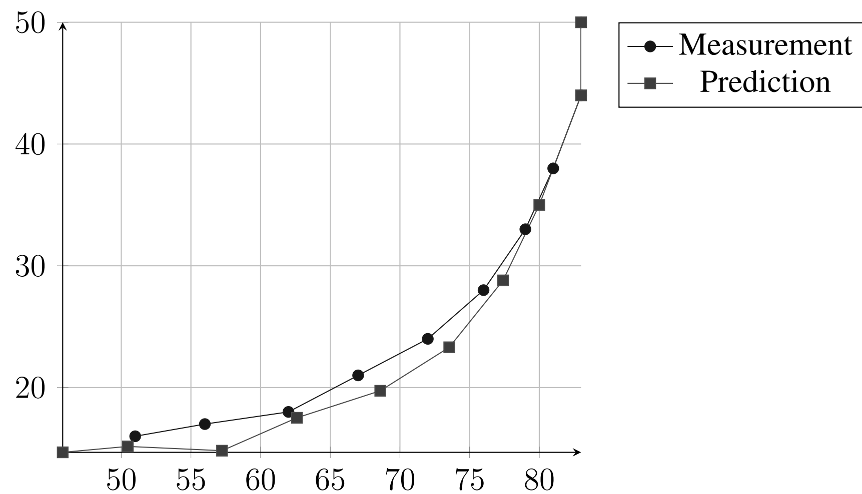


Figure 7.5: Kalman filter predicting object that is moving in curve moving from right to left

7.4 Simple Online and Realtime Tracking

In 2016 Bewley et al. [54] proposed a tracker algorithm called Simple Online and Realtime Tracking (SORT). SORT is an online tracker and focuses on frame-to-frame associations. Unlike many other detectors in 2015, SORT does not try to be robust to detection errors . Bewley et al. tells in their paper that SORT was motivated by the *MOTChal-*

lenge released in 2015. In *MOTChallenge's* 15 data set results, SORT was the best online tracking algorithm in 2015 with the MOTA of 33.4 whereas the highest MOTA for batch trackers was 33.7. [54]

SORT is a pragmatic tracking algorithm and is based on two classical methods. In its core SORT uses *Kalman filter* (explained in earlier chapter 7.3) and *Hungarian algorithm* (explained in earlier in the end of chapter 7.2). For every trackable object SORT keeps a state for each object to work as an estimation model:

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

In the state model, u and v represents the objects centroid's location in horizontal and vertical location. s represents the scale and r is the aspect ratio of objects bounding box. When SORT associates a new detection to the target object, the new bounding box is used to update the target's state. With this update, Kalman filter updates the velocity component of the model. [54]

To handle data association, SORT uses *Hungarian algorithm* with *Intersection over union*. For existing targets, new bounding box location is estimated for current frame by using Kalman filter for predicting the location. Then an assignment cost matrix is calculated for each detection and estimated bounding box as a distance of IoU. Assignment matrix is solved with Hungarian algorithm, but the assignments can be rejected if the detection to target overlap is less than IOU_{min} given as a hyperparameter. According to Bewley et al. IoU distance handles well short-term occlusions happening when objects pass each others. [54]

SORT creates a new tracking if every overlap for detection is less than IoU_{min}. New trackers velocity is set to zero because it is not observed at this point. The tracker goes through what Bewley et al. calls *probationery period* [54]. During this period tracker need to be associated with enough detections as an evidence to prevent tracking of false positives. [54]

SORT creates and maintains several trackers. In order to prevent an unbounded growth

of the number of trackers, SORT will remove trackers. Every tracker keeps a track of the number of consecutive non-detected frames. SORT then terminates the tracker if the number of frames for tracker is larger than T_{Lost}. Removing trackers also helps SORT to prevent localisation errors which may be caused by trackers' predictions. Bewley et al. used $T_{Lost} = 1$ [54] in all their experiences with SORT. According to them, this value has two reasons. The used constant velocity model in SORT is a poor predictor and they created SORT to solve frame-to-frame tracking. [54]

7.5 Sort with Deep Association Metric

SORT approaches the MOT problem as a frame-by-frame problem. Because of this and its dependency on detectors' ability to detect objects well. In Bewley et al. paper [54] has the highest number of identification switches (1001) when comparing to other online trackers. In 2017 Wojke et al. [55] proposed a new version of SORT that uses Convolutional neural network (CNN) to work with the Kalman filter to reduce the number of identification switches. Sort with Deep Association Metric is better known as *Deep SORT*. [55]

According to the authors of Deep SORT [55], this tracker's track handling and state estimation are mostly identical to SORT. Deep SORT's state space for Kalman filter is eight dimensional [55]:

$$(u, v, \gamma, h, \dot{x}, \dot{y}, \dot{\gamma}, \dot{h})$$

Just like SORT, Deep SORT uses Hungarian algorithm to solve assignment problem for associating trackers with detections. In order to use motion information when associating detections with trackers, Deep SORT uses Mahalanobis distance[55][56] for Kalman predictions and new detections. Mahalanobis distance is almost like Euclidean distance because both can be used to give a distance between two points[56]. In Mahalanobis' case, the given distance measures how far is the point from a distribution. Mahalanobis distance is used in Deep SORT to measure how far away a detection is from a track location by

using Kalman filter's predicted state[55]. Because predictions provided by Kalman filter are rough estimates, Deep SORT adds another metric called Deep Appearance Descriptor.

Every other tracker described earlier can do tracking with just only detections' bounding boxes. This is where Deep SORT differentiates from others. Deep SORT needs the image where the provided detections where extracted. Reason for this is the CNN in Deep SORT. Wojke et al. calls this CNN as *Deep Appearance Descriptor* and it is used to extract 128 values as a feature vector from the image within the bounding box. In table 7.1 we can see Deep SORT's CNN architecture for Deep Appearance Descriptor. Deep SORT's CNN is a wide residual network[55] with two convolutional layers, max pooling layer and six residual layers, dense layer and batch layer with L2 normalization layer. Dense layer is used to compute the global feature map of 128 values and the batch and L2 normalization layer is used to convert values to be compatible with the used cosine appearance metric. [55] Nicolai Wojke et al. used Motion Analysis and Re-identification Set (MARS) data set to train the CNN to extract features. [55] MARS data set is a large data set containing multiple videos used to research person reidentification and it contains 1,261 different pedestrians [57].

Index	Layer type	Patch Size	Stride Size	Output Size
1	Convolutional	3×3	1	$32 \times 128 \times 64$
2	Convolutional	3×3	1	$32 \times 128 \times 64$
3	Max Pool	3×3	2	$32 \times 64 \times 32$
4	Residual	3×3	1	$32 \times 64 \times 32$
5	Residual	3×3	1	$32 \times 64 \times 32$
6	Residual	3×3	2	$64 \times 32 \times 16$
7	Residual	3×3	1	$64 \times 32 \times 16$
8	Residual	3×3	2	$128 \times 16 \times 8$
9	Residual	3×3	1	$128 \times 16 \times 8$
10	Dense			128 x 1
11	Batch and L2 normalization			128 x 1

Table 7.1: Deep SORT's CNN architecture [55]

Chapter 8

Results

8.1 Object detectors

Comparing different object detectors is no small task. Training and fine-tuning different object detection models are very time consuming and it can take very long time. One of the most used data set for object detectors is called Common Objects in Context (COCO) provided by Microsoft [58]. Many of the state-of-the-art object detection models are trained and tested with COCO data set [58] and their results are given in COCO's own metric *mAP@[.5, .95]*. This metric shows how well does the model perform over multiple different categories with different accuracies.

Both neural network based object detectors are very capable detectors. Faster R-CNN achieved mAP@[.5, .95] score of 21.9 [40] and one of the latest invariants of Faster R-CNN called as *Faster R-CNN w TDM* got 36.8[43] as a score. YOLOv3 does also well for COCO's evaluation with a mAP@[.5, .95] of 33.0[43]. For Histogram of Oriented Gradients with Support Vector Machine, I was unable to find COCO's mAP@[.5, .95] results. This might be so, because object detection field has been dominated by neural networks for a long time [14].

8.2 Tracking

When comparing different trackers' performance, I used precalculated detections provided in *MOT20* data set [18]. I decided to use training data set's precalculated detections and ground-truths to evaluate how well does the trackers work. Training data set contains four (4) different video sequences with different lenghts. These four sequences are described in table 8.1.

Sequence	Frames	Tracks	Density
MOT20-01	429	74	46.32
MOT20-02	2782	270	55.62
MOT20-03	2405	702	130.42
MOT20-05	3315	1169	194.98

Table 8.1: MOT20 training sequences' data [18]

Method	MOTA	MOTP	IDF1	FM	ID Switch
Euclidean distance	16.4	**87.6**	3.8	53802	393360
Kalman filter	50.3	87.2	16.7	35793	46593
SORT	52.0	87.5	**52.1**	40489	**5809**
Deep SORT	**53.5**	86.2	43.5	**24711**	8228

Table 8.2: Results of Tracking with provided detections in MOT20 [18]

Evaluation was done with *MOTChallenge's* official evaluation kit **TrackEval** [59]. Result files were generated from the provided detections files for sequences *MOT20-01*, *MOT20-02*, *MOT20-03* and *MOT20-05*. Each of the result files were given to the evaluation script and given results are shown in the table 8.2. Every tested tracker was used as an online tracker, so each of the trackers were initialized and detections were fed to trackers

frame-by-frame. Result files were created during the runtime from trackers' returned IDs for different bounding boxes.

Tracker using Euclidean distance did not do very well compared to other trackers. MOTChallenge provides very challenging sequences with very high density of people. This caused tracker to switch ids almost between every frame. After manually inspecting tracker's result file, I found out that Euclidean distance worked well for objects if the bounding boxes did not overlap. Many of the ID switches did not make much sense in multiple cases, but I assume that it is causes by the used algorithm to solve assignment problem. It might be because Euclidean distance is not great to be used with Hungarian algorithm. I used my own implementation of Euclidean distance-based tracker which can be seen in appendix A.

Sequence	**MOTA**	**MOTP**	**IDF1**	**FM**	**ID Switch**
MOT20-01	14.5	91.1	6.6	849	7706
MOT20-02	24.7	91.9	7.7	3603	41905
MOT20-03	20.9	85.8	4.4	14188	97662
MOT20-05	12.2	82.3	2.5	35162	246087
Combined	16.4	87.6	3.8	53802	393360

Table 8.3: Results for Euclidean distance based tracker with provided detections in MOT20 [18]

Kalman filter-based tracker did much better than the Euclidean based tracker as could be expected. Using the state estimation where trackable object would be in the next frame helped the tracker to handle cases where object was not detected by the detector. This led to having only 46593 identification switches compared to Euclidean's 393360. One surprising result is that this tracker scored better than SORT for the MOTA score. For testing, I created and used my own implementation of Kalman filter tracker. It uses

Kalman filter to predict trackable object's new location and then calculates the Euclidean distances between every detection and predictions. Most optimal associations are done by Hungarian algorithm, implementation of Kalman based tracker can bee seen in appendix B.

Sequence	MOTA	MOTP	IDF1	FM	ID Switch
MOT20-01	57.6	90.6	42.5	279	328
MOT20-02	52.8	91.4	28.3	1946	2475
MOT20-03	49.5	85.4	16.6	11340	14277
MOT20-05	49.9	86.9	13.1	22228	29513
Combined	50.3	87.2	16.7	35793	46593

Table 8.4: Results for Kalman filter based tracker with provided detections in MOT20 [18]

Simple Online and Realtime Tracking (SORT) did very well compared to Euclidean distance and Kalman filter-based trackers. SORT has the lowest number of ID switches, fragmentation and Identification F1 Score. Because SORT has a probationary period for new detections, this tracker does not provide any tracking for the first two frames. This reduces MOTA and MOTP scores because TrackEval compares result files for the ground-truth files. When manually inspecting SORT's results I did not find anything weird, as a matter of fact it handled detections in dense areas very well. To test SORT, I used its official open-source implementation provided in its official GitHub repository (`https://github.com/abewley/sort`)[54] and implementation's structure was used as an inspiration for my implementation for Euclidean and Kalman trackers. For testing SORT I used non-default values for T_{Lost} and time for probationary period. T_{Lost} was 100 and probationary period was set to 1. I selected these values because SORT had submitted its new results for MOT20 challenge with those values for *MOTChallenge*'s official result board (`https://motchallenge.net/results/MOT20/`)

[1]. Change from default values to new values caused MOTA to increase from 45.8 to 50.3 and mostly made results better. Only negative effect was that the *FM* increased from 19570 to 46593 which is significantly worse.

Sequence	MOTA	MOTP	IDF1	FM	ID Switch
MOT20-01	56.4	91.5	58.8	368	77
MOT20-02	52.3	92.0	52.2	2459	516
MOT20-03	51.2	85.7	56.1	12941	2176
MOT20-05	52.2	87.3	49.9	24721	3040
Combined	52.0	87.5	52.1	40489	5809

Table 8.5: Results for SORT with provided detections in MOT20 [18]

Sort with Deep Association Metric (Deep SORT) got the highest Multi-Object Tracking Accuracy (MOTA) which is usually the primary metric used to compare different trackers and their performance. For testing Deep SORT, I used its official open-source implementation found at its GitHub repository (`https://github.com/nwojke/deep_sort`)[55]. Because Deep SORT was going to be used for tracking pedestrians from video sequences, I decided to use the pre-trained CNN provided by original authors [55]. Reason for this decision was that the provided pre-trained CNN was trained on MARS data set and was therefore capable for extracting features of different pedestrians. In the original paper of Deep SORT [55], the difference between SORT's and Deep SORT's MOTA was 1.6 for *MOT16*[25] data set. In my testing the difference between these two trackers was 1.5 for *MOT20* training data set. It is interesting to see that even with a different data set, the different between SORT and DeepSORT remains almost the same. Other interesting thing can be found in the trackers' result table 8.2. SORT's number of identification switches is almost half of the number for Deep SORT. In the Deep SORT's paper [55] authors says that Deep Appearance Descriptor would lower the

number of Identification switches compared to the SORT.

Sequence	**MOTA**	**MOTP**	**IDF1**	**FM**	**ID Switch**
MOT20-01	56.6	90.1	54.3	242	112
MOT20-02	52.6	90.7	45.0	1552	766
MOT20-03	53.3	84.1	46.6	8129	2711
MOT20-05	53.7	86.1	41.3	14788	4639
Combined	53.5	86.2	43.5	24711	8228

Table 8.6: Results for Deep SORT with provided detections in MOT20 [18]

Chapter 9

Conclusion

According to different papers' results and reviews, I think that we cannot justify the replacement of neural network-based detector with non-neural network detector. If the goal is to detect object from a single class, Histogram of Oriented Gradients might be a reasonable solution, but in my opinion, we should use neural network-based detector. It would be a good research idea to do extensive research on HOG and its variants.

Also, because trackers' performance is highly correlated with detections quality, I think that it is required to have well performing detector like YOLO V3, Faster R-CNN or some other object detection model. When selecting the object detector, selection should be done regarding the planned usage. Many two-stage detectors like Faster R-CNN are usually more accurate than single-stage detectors, but are not capable for real-time detections. For extracting detections, for example from video, two-stage detectors might be the reasonable option. If the goal is to use real-time video feed, like from a traffic camera, single-stage detectors are almost mandatory.

For tracking algorithm, I think that it is reasonable to replace deep learning-based trackers with more classical methods. I inspected the state-of-the-art tracking algorithms from MOTChallenge 2020 results [18]. I found out that to achieve MOTA higher than SORT or DeepSORT, it would have a great impact for the time needed to associate detections with old ones. For example, SORT is capable to process around 57 frames per

second (FPS) according to the MOTChallenge 2020 results. If we would use state-of-the-art tracking algorithm, the FPS would reduce to below seven FPS. In my opinion this has no place in real life if we are trying to do real-time tracking. If it is possible to spend that much time per frame, I think that using batch tracking algorithm would be reasonable.

From tables 8.2, 8.5 and 8.6 we can see that the difference between SORT and Deep SORT, Multi-Object Tracking Accuracy (MOTA) scores are not far apart. Because MOTA scores are so close to each others, I think that SORT should be used instead of DeepSORT in this scenario. The main difference between SORT and Deep SORT is that Deep SORT uses a CNN model to help with tracings. SORT and Deep SORT were both state-of-the-art tracking algorithms, but in this case SORT was able to generalise tracking better than Deep SORT. CNN model in Deep SORT has precalculated weights that the author [55] provides and with them, model should be able to extract features well from pedestrians. If we need to track something else than pedestrians, we will have to train Deep SORT's CNN model for any other category.

References

[1] Patrick Dendorfer, Aljosa Osep, Anton Milan, Konrad Schindler, Daniel Cremers, Ian Reid, Stefan Roth and Laura Leal-Taixé. Motchallenge: A benchmark for single-camera multiple target tracking. *International Journal of Computer Vision*, 129(4):845–881, 2021.

[2] Kevin Gurney. *An introduction to neural networks*. CRC press, 1997.

[3] Stephen Marsland. *Machine learning: an algorithmic perspective*. CRC press, 2015.

[4] Hamid Karimi, Tyler Derr and Jiliang Tang. Characterizing the decision boundary of deep neural networks. *arXiv preprint arXiv:1912.11460*, 2019.

[5] Wlodzislaw Duch and Norbert Jankowski. Survey of neural transfer functions. *Neural Computing Surveys*, 2(1):163–212, 1999.

[6] Prajit Ramachandran, Barret Zoph and Quoc V Le. Searching for activation functions. *arXiv preprint arXiv:1710.05941*, 2017.

[7] Ian Goodfellow, Yoshua Bengio and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[8] Saad Albawi, Tareq Abed Mohammed and Saad Al-Zawi. *2017 INTERNATIONAL CONFERENCE ON ENGINEERING AND TECHNOLOGY (ICET)* Understanding of a convolutional neural network, pages 1–6. Ieee, 2017.

[9] Kaiming He, Xiangyu Zhang, Shaoqing Ren and Jian Sun. *PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* Deep residual learning for image recognition, pages 770–778. 2016.

[10] Sergey Ioffe and Christian Szegedy. *INTERNATIONAL CONFERENCE ON MACHINE LEARNING* Batch normalization: Accelerating deep network training by reducing internal covariate shift, pages 448–456. PMLR, 2015.

[11] Yann Le Cun, Lionel D Jackel, Brian Boser, John S Denker, Henry P Graf, Isabelle Guyon, Don Henderson, Richard E Howard and William Hubbard. Handwritten digit recognition: Applications of neural network chips and automatic learning. *IEEE Communications Magazine*, 27(11):41–46, 1989.

[12] Alex Krizhevsky, Ilya Sutskever and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[13] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272, 1968.

[14] Zhengxia Zou, Zhenwei Shi, Yuhong Guo and Jieping Ye. Object detection in 20 years: A survey. *arXiv preprint arXiv:1905.05055*, 2019.

[15] Yann LeCun, Yoshua Bengio and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[16] Sertac Karaman, Ariel Anders, Michael Boulet, Jane Connor, Kenneth Gregson, Winter Guerra, Owen Guldner, Mubarik Mohamoud, Brian Plancher, Robert Shin et al. *2017 IEEE INTEGRATED STEM EDUCATION CONFERENCE (ISEC)* Project-based, collaborative, algorithmic robotics for high school students: Programming self-driving race cars at MIT, pages 195–203. IEEE, 2017.

[17] Jianfeng Ren, Nasser Kehtarnavaz and Leonardo Estevez. *2008 IEEE DALLAS CIRCUITS AND SYSTEMS WORKSHOP: SYSTEM-ON-CHIP-DESIGN, APPLICATIONS, INTEGRATION, AND SOFTWARE* Real-time optimization of Viola-Jones face detection for mobile platforms, pages 1–4. IEEE, 2008.

[18] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler and L. Leal-Taixé. MOT20: A benchmark for multi object tracking in crowded scenes. *arXiv:2003.09003[cs]*, 2020. URL `http://arxiv.org/abs/1906.04567`, arXiv: 2003.09003.

[19] Alexander Kirillov, Kaiming He, Ross Girshick, Carsten Rother and Piotr Dollár. *PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* Panoptic segmentation, pages 9404–9413. 2019.

[20] Rasmus Rothe, Matthieu Guillaumin and Luc Van Gool. *ASIAN CONFERENCE ON COMPUTER VISION* Non-maximum suppression for object detection by passing messages between windows, pages 290–306. Springer, 2014.

[21] David L Olson and Dursun Delen. *Advanced data mining techniques*. Springer Science & Business Media, 2008.

[22] Hamid Rezatofighi, Nathan Tsoi, JunYoung Gwak, Amir Sadeghian, Ian Reid and Silvio Savarese. *PROCEEDINGS OF THE IEEE/CVF CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* Generalized intersection over union: A metric and a loss for bounding box regression, pages 658–666. 2019.

[23] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[24] Laura Leal-Taixé, Anton Milan, Ian D. Reid, Stefan Roth and Konrad Schindler. MOTChallenge 2015: Towards a Benchmark for Multi-Target Tracking. *CoRR*, abs/1504.01942, 2015. URL `http://arxiv.org/abs/1504.01942`.

[25] A. Milan, L. Leal-Taixé, I. Reid, S. Roth and K. Schindler. MOT16: A Benchmark for Multi-Object Tracking. *arXiv:1603.00831 [cs]*, 2016. URL `http://arxiv.org/abs/1603.00831`, arXiv: 1603.00831.

[26] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, 2008:1–10, 2008.

[27] Ergys Ristani, Francesco Solera, Roger Zou, Rita Cucchiara and Carlo Tomasi. *EUROPEAN CONFERENCE ON COMPUTER VISION* Performance measures and a data set for multi-target, multi-camera tracking, pages 17–35. Springer, 2016.

[28] Bo Wu and Ram Nevatia. *2006 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'06)* Tracking of multiple, partially occluded humans based on static body part detection, volume 1, pages 951–958. IEEE, 2006.

[29] Laura Leal-Taixé. Multiple object tracking with context awareness. 11 2014.

[30] Xin Lu, Quanquan Li, Buyu Li and Junjie Yan. MimicDet: Bridging the Gap Between One-Stage and Two-Stage Object Detection. *arXiv preprint arXiv:2009.11528*, 2020.

[31] Navneet Dalal and Bill Triggs. *2005 IEEE COMPUTER SOCIETY CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION (CVPR'05)* Histograms of oriented gradients for human detection, volume 1, pages 886–893. IEEE, 2005.

[32] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[33] William S Noble. What is a support vector machine? *Nature biotechnology*, 24(12):1565–1567, 2006.

[34] K Mike Tao. *PROCEEDINGS OF 27TH ASILOMAR CONFERENCE ON SIGNALS, SYSTEMS AND COMPUTERS* A closer look at the radial basis function (RBF) networks, pages 401–405. IEEE, 1993.

[35] Jasper RR Uijlings, Koen EA Van De Sande, Theo Gevers and Arnold WM Smeulders. Selective search for object recognition. *International journal of computer vision*, 104(2):154–171, 2013.

[36] Xianfang Zeng, Wenxuan Wu, Guangzhong Tian, Fuxin Li and Yong Liu. Deep Superpixel Convolutional Network for Image Recognition. *IEEE Signal Processing Letters*, 28:922–926, 2021.

[37] Ross Girshick, Jeff Donahue, Trevor Darrell and Jitendra Malik. *PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* Rich feature hierarchies for accurate object detection and semantic segmentation, pages 580–587. 2014.

[38] Chunhui Gu, Joseph J Lim, Pablo Arbeláez and Jitendra Malik. *2009 IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* Recognition using regions, pages 1030–1037. IEEE, 2009.

[39] Ross Girshick. *PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION* Fast r-cnn, pages 1440–1448. 2015.

[40] Shaoqing Ren, Kaiming He, Ross Girshick and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[41] Joseph Redmon, Santosh Divvala, Ross Girshick and Ali Farhadi. *PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* You only look once: Unified, real-time object detection, pages 779–788. 2016.

[42] Joseph Redmon and Ali Farhadi. *PROCEEDINGS OF THE IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION* YOLO9000: better, faster, stronger, pages 7263–7271. 2017.

[43] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[44] Yu Xiang, Alexandre Alahi and Silvio Savarese. *PROCEEDINGS OF THE IEEE INTERNATIONAL CONFERENCE ON COMPUTER VISION* Learning to track: Online multi-object tracking by decision making, pages 4705–4713. 2015.

[45] Jerome Berclaz, Francois Fleuret, Engin Turetken and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE transactions on pattern analysis and machine intelligence*, 33(9):1806–1819, 2011.

[46] Harold W Kuhn. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[47] James Munkres. Algorithms for the assignment and transportation problems. *Journal of the society for industrial and applied mathematics*, 5(1):32–38, 1957.

[48] R. E. Kalman. A New Approach to Linear Filtering and Prediction Problems. *Journal of Basic Engineering*, 82(1):35–45, 03 1960. ISSN 0021-9223. URL https://doi.org/10.1115/1.3662552.

[49] Mohinder S Grewal and Angus P Andrews. Applications of Kalman filtering in aerospace 1960 to the present [historical perspectives]. *IEEE Control Systems Magazine*, 30(3):69–78, 2010.

[50] Susmita Bhattacharyya, Dinesh L Mute and Demoz Gebre-Egziabher. 0363 *AIAA SCITECH 2019 FORUM* Kalman Filter-Based Reliable GNSS Positioning for Aircraft Navigation. 2019.

[51] Youngjoo Kim and Hyochoong Bang. Introduction to Kalman filter and its applications. *Introduction and Implementations of the Kalman Filter*, 1:1–16, 2018.

[52] J Morris. The Kalman filter: A robust estimator for some classes of linear quadratic problems. *IEEE Transactions on Information Theory*, 22(5):526–534, 1976.

[53] Ramsey Faragher. Understanding the basis of the kalman filter via a simple and intuitive derivation [lecture notes]. *IEEE Signal processing magazine*, 29(5):128–132, 2012.

[54] Alex Bewley, Zongyuan Ge, Lionel Ott, Fabio Ramos and Ben Upcroft. *2016 IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP)* Simple online and realtime tracking, pages 3464–3468. IEEE, 2016.

[55] Nicolai Wojke, Alex Bewley and Dietrich Paulus. *2017 IEEE INTERNATIONAL CONFERENCE ON IMAGE PROCESSING (ICIP)* Simple online and realtime tracking with a deep association metric, pages 3645–3649. IEEE, 2017.

[56] Roy De Maesschalck, Delphine Jouan-Rimbaud and Désiré L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[57] Springer. *MARS: A Video Benchmark for Large-Scale Person Re-identification*, 2016.

[58] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár and C Lawrence Zitnick. *EUROPEAN CONFERENCE ON COMPUTER VISION* Microsoft coco: Common objects in context, pages 740–755. Springer, 2014.

[59] Arne Hoffhues Jonathon Luiten. TrackEval. https://github.com/JonathonLuiten/TrackEval, 2020.

Appendix A

Naive object tracker using Euclidean distance

```
1  import numpy as np
2  import math
3  from scipy.optimize import linear_sum_assignment
4
5
6  class EuclideanTracker(object):
7      count = 0
8
9      def __init__(self, bb):
10         self.id = EuclideanTracker.count
11         EuclideanTracker.count += 1
12         self.last_location = EuclideanTracker.bounding_box_to_centroid(
               bb)
13         self.bb = bb
14
15     def update(self, bb):
16         self.last_location = EuclideanTracker.bounding_box_to_centroid(
               bb)
17         self.bb = bb
```

```
18
19      def get_state(self):
20          return self.bb
21
22      def bounding_box_to_centroid(bb):
23          return np.array([(bb[0] + bb[2]) / 2, (bb[1] + bb[3]) / 2])
24
25      def __repr__(self):
26          return str(self.id) + ' ' + str(self.last_location)
27
28
29  class Euclidean(object):
30
31      def __init__(self, max_distance=1500):
32          self.max_distance = max_distance
33          self.trackers = []
34          self.frame_count = 0
35
36      def __calculate_distance_matrix(self, detections, trackers):
37          distance_matrix = np.zeros((len(detections), len(trackers)))
38          for i in range(len(detections)):
39              for j in range(len(trackers)):
40                  distance_matrix[i][j] = self.
                        __calculate_euclidean_distance(
41                      EuclideanTracker.bounding_box_to_centroid(
42                          detections[i]),
43                      trackers[j]
44                  )
45          return distance_matrix
46
47      def __calculate_euclidean_distance(self, point_a, point_b):
48          return math.sqrt((point_a[0] - point_b[0])**2 + (point_a[1] -
                point_b[1])**2)
```

```
49
50      def __get_detections_to_trackers_associations(self, detections,
            trackers):
51          if len(trackers) == 0:
52              return np.empty((0, 2), dtype=int), np.arange(len(
                    detections)), np.empty((0, 2), dtype=int)
53
54          distance_matrix = self.__calculate_distance_matrix(
55              detections, trackers)
56          detection_row, tracker_column = linear_sum_assignment(
                distance_matrix)
57          matches = np.array(list(zip(detection_row, tracker_column)))
58          matched_indices = []
59          unmatched_detections = []
60          unmatched_trackers = []
61          if len(matches[0]) > 0:
62              for i in range(len(matches)):
63                  detection_index = matches[i][0]
64                  tracker_index = matches[i][1]
65                  distance = distance_matrix[detection_index][
                        tracker_index]
66                  if(distance > self.max_distance):
67                      unmatched_trackers.append(tracker_index)
68                      unmatched_detections.append(detection_index)
69                  else:
70                      matched_indices.append(matches[i, :])
71              matched_indices = np.array(matched_indices)
72          else:
73              matched_indices = np.empty(shape=(0, 2))
74
75          for d, det in enumerate(detections):
76              if(len(matched_indices) > 0 and d not in matched_indices[:,
                    0]):
```

```
77                 unmatched_detections.append(d)
78
79         for t, trk in enumerate(trackers):
80             if(len(matched_indices) > 0 and t not in matched_indices[:,
                   1]):
81                 unmatched_trackers.append(t)
82
83         return matched_indices, np.array(unmatched_detections), np.
               array(unmatched_trackers)
84
85     def update(self, dets=np.empty((0, 2))):
86         self.frame_count += 1
87         trks = np.zeros((len(self.trackers), 2))
88         ret = []
89         for t, trk in enumerate(trks):
90             pos = self.trackers[t].get_state()
91             trk[:] = [pos[0], pos[1]]
92         matched, unmatched_dets, unmatched_trks = self.
               __get_detections_to_trackers_associations(
93             dets, trks)
94
95         for m in matched:
96             self.trackers[m[1]].update(dets[m[0], :])
97
98         for i in unmatched_dets:
99             trk = EuclideanTracker(dets[i, :])
100            self.trackers.append(trk)
101
102        for m in reversed(sorted(unmatched_trks)):
103            self.trackers.pop(m)
104
105        for trk in self.trackers:
106            ret.append(np.concatenate(
```

```
107                     (trk.get_state(), [trk.id+1])).reshape(1, -1))
108
109         if(len(ret) > 0):
110             return np.concatenate(ret)
111         return np.empty((0, 2))
```

Appendix B

Naive object tracker using Kalman filter

```
1  import numpy as np
2  import math
3  from scipy.optimize import linear_sum_assignment
4  from filterpy.kalman import KalmanFilter
5
6
7  class KalmanFilterTracker(object):
8      count = 0
9
10     def __init__(self, bb, max_age=5):
11         self.id = KalmanFilterTracker.count
12         KalmanFilterTracker.count += 1
13         self.max_age = max_age
14         self.number_of_predicted_concurrent_frames = 0
15         self.bb = bb
16         self.kf = KalmanFilter(dim_x=4, dim_z=2)
17         self.kf.F = np.array(
18             [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])
19         self.kf.H = np.array([[1, 0, 0, 0], [0, 1, 0, 0]])
20
21         self.kf.R[2:, 2:] *= 10.
```

```
22          self.kf.P[4:, 4:] *= 1000.
23          self.kf.P *= 10.
24          self.kf.Q[-1, -1] *= 0.01
25          self.kf.Q[4:, 4:] *= 0.01
26          self.kf.x[:2] = self.as_centroid()
27
28      def update(self, bb):
29          self.number_of_predicted_concurrent_frames = 0
30          self.bb = bb
31          self.kf.update(self.as_centroid())
32
33      def get_state(self):
34          return self.bb
35
36      def as_centroid(self):
37          x = (self.bb[0] + self.bb[2]) / 2
38          y = (self.bb[1] + self.bb[3]) / 2
39          return np.array([[np.float32(x)], [np.float32(y)]])
40
41      def predict(self):
42          self.kf.predict()
43          return self.kf.x
44
45      def bounding_box_to_centroid(bb):
46          x = (bb[0] + bb[2]) / 2
47          y = (bb[1] + bb[3]) / 2
48          return np.array([[np.float32(x)], [np.float32(y)]])
49
50      def update_without_real_detection(self):
51          self.number_of_predicted_concurrent_frames += 1
52          self.kf.predict()
53          return self.kf.x
54
```

```
55      def __repr__(self):
56          return str(self.id) + ' ' + str(self.bb)
57
58
59  class Kalman(object):
60
61      def __init__(self, max_age):
62          self.trackers = []
63          self.frame_count = 0
64          self.max_age = max_age
65
66      def __calculate_distance_matrix(self, detections, trackers):
67          distance_matrix = np.zeros((len(detections), len(trackers)))
68          for i in range(len(detections)):
69              for j in range(len(trackers)):
70                  distance_matrix[i][j] = self.
                        __calculate_euclidean_distance(
71                      KalmanFilterTracker.bounding_box_to_centroid(
                            detections[i]), trackers[j])
72          return distance_matrix
73
74      def __calculate_euclidean_distance(self, point_a, point_b):
75          eucl_dist = math.sqrt(
76              (point_a[0] - point_b[0])**2 + (point_a[1] - point_b[1])
                    **2)
77          return eucl_dist
78
79      def __get_detections_to_trackers_associations(self, detections,
              trackers):
80          if len(trackers) == 0:
81              return np.empty((0, 2), dtype=int), np.arange(len(
                    detections)), np.empty((0, 2), dtype=int)
82          i = len(self.trackers) - 1
```

```
83          for trk in reversed(self.trackers):
84              if trk.number_of_predicted_concurrent_frames > self.max_age
                    :
85                  self.trackers.pop(i)
86
87          trackers_preds = np.zeros((len(self.trackers), 2))
88          for tracker, trk in zip(self.trackers, trackers_preds):
89              pos = tracker.predict()
90              trk[:] = [pos[0], pos[1]]
91          distance_matrix = self.__calculate_distance_matrix(
92              detections, trackers_preds)
93          detection_row, tracker_column = linear_sum_assignment(
                distance_matrix)
94          matches = np.array(list(zip(detection_row, tracker_column)))
95          matched_indices = []
96          unmatched_detections = []
97          unmatched_trackers = []
98          if len(matches) > 0 and len(matches[0]) > 0:
99              for i in range(len(matches)):
100                 detection_index = matches[i][0]
101                 tracker_index = matches[i][1]
102                 distance = distance_matrix[detection_index][
                        tracker_index]
103                 matched_indices.append(matches[i, :])
104             matched_indices = np.array(matched_indices)
105         else:
106             matched_indices = np.empty(shape=(0, 2))
107
108         for d, det in enumerate(detections):
109             if(len(matched_indices) > 0 and d not in matched_indices[:,
                    0]):
110                 unmatched_detections.append(d)
111
```

```
112         for t, trk in enumerate(trackers):
113             if(len(matched_indices) > 0 and t not in matched_indices[:,
                    1]):
114                 unmatched_trackers.append(t)
115 
116         return matched_indices, np.array(unmatched_detections), np.
                array(unmatched_trackers)
117 
118     def update(self, dets=np.empty((0, 2))):
119         self.frame_count += 1
120         trks = np.zeros((len(self.trackers), 2))
121         to_del = []
122         ret = []
123         for t, trk in enumerate(trks):
124             pos = self.trackers[t].get_state()
125             trk[:] = [pos[0], pos[1]]
126         matched, unmatched_dets, unmatched_trks = self.
                __get_detections_to_trackers_associations(
127             dets, trks)
128 
129         for m in matched:
130             self.trackers[m[1]].update(dets[m[0], :])
131 
132         for i in unmatched_dets:
133             trk = KalmanFilterTracker(dets[i, :])
134             self.trackers.append(trk)
135         for m in reversed(sorted(unmatched_trks)):
136             if m < len(self.trackers):
137                 self.trackers[m].update_without_real_detection()
138 
139         for trk in self.trackers:
140             ret.append(np.concatenate(
141                 (trk.get_state(), [trk.id+1])).reshape(1, -1))
```

```
142
143         if(len(ret) > 0):
144             return np.concatenate(ret)
145         return np.empty((0, 2))
```