FACULTY OF INFORMATION TECHNOLOGY CTU IN PRAGUE

Assignment of bachelor's thesis

Title:	Algorithms for video analysis of customer behavior in front of retail store
Student:	David Mašek
Supervisor:	Ing. Lukáš Brchl
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

Cílem práce je návrh a implementace algoritmů, jejichž cílem je umožnit detekovat a sledovat osoby ve videozáznamu a pomocí extrakce obrazových charakteristik jim vytvořit unikátní identitu. Kromě toho je součástí zadání vytěžování užitečných informací o zákaznících, které je možné využít v retailu (např. demografické údaje, časové a poziční data).

- Proveďte rešerši existujících řešení.
- Proveďte návrh a implementaci algoritmů detekce a re-identifikace s využitím algoritmů počítačového vidění.
- Prozkoumejte a implementujte vhodné algoritmy pro vytěžování užitečných informací o osobách pro využití v retailu.
- Zvažte možnost vytvoření vlastního datasetu pro vyhodnocení algoritmů nebo vyberte z existujících.
- Proveďte zhodnocení dosažených výsledků a navrhněte budoucí rozšíření.

Electronically approved by Ing. Karel Klouda, Ph.D. on 11 February 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Bachelor's thesis

Algorithms for video analysis of customer behavior in front of retail store

David Mašek

Department of Applied Mathematics
Supervisor: Ing. Lukáš Brchl

May 13, 2021

Acknowledgements

I would like to thank my supervisor Ing. Lukáš Brchl for his guidance and advice. Furthermore, I would like to thank the ImproLab team at FIT CTU for providing me with the thesis topic. Finally, I wish to thank my friends (with a special mention to members of CHS), family and girlfriend for their support.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis.

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended, in particular that the Czech Technical University in Prague has the right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act.

In Prague on May 13, 2021 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Mašek, David. *Algorithms for video analysis of customer behavior in front of retail store*. Bachelor's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2021.

Abstract

This thesis aims to design a framework for tracking people based on a stream from a single stationary camera, with the secondary goal of extracting age and gender information for tracked people. The focus of this work is on the retail shop environment. The main algorithm follows the tracking by detection approach. The matching of detections to tracks is done based on spatial and visual information from convolutional neural networks. Kalman filter is used for robust state representation and updates. We evaluate the algorithm with multiple detector models on a dataset collected from the target environment. We also evaluate the performance improvements from using the TensorRT optimization framework. The resulting application achieves 0.91 MOTA on the testing dataset, with frame rate of 13 FPS on the Jetson NX platform.

Keywords computer vision, people tracking, demographic information extraction, TensorRT

Abstrakt

Cílem této práce je návrh frameworku pro sledování osob na záznamu z jedné staticky umístěné kamery, s vedlejším cílem extrakce věku a pohlaví sledovaných osob. Práce je zaměřena na prostředí maloobchodu. Hlavní algoritmus funguje na principu sledování na základě detekcí. Asociace detekcí k identitám je založena na informacích o poloze a vzhledu získaných z konvolučních neuronových sítí. Kalman filtr je použit pro robustní reprezentaci identit a jejich aktualizaci. Algoritmus vyhodnocujeme s několika modely pro detekci na datasetu získaném z cílového prostředí. Také vyhodnocujeme zlepšení výkonu zíkané použitím optimalizačního frameworku TensorRT. Výsledná aplikace dosahuje 0.91 MOTA na testovacím datasetu, se snímkovací frekvencí 13 snímků za sekundu na zařízení Jetson NX.

Klíčová slova počítačové vidění, sledování osob, extrakce demografických údajů, TensorRT

Contents

List of Figures

Introduction

The topic of this thesis is automatic video analysis with the goal of tracking people and creating unique identities for them, including demographic information such as age and gender. Movement and demographics can be a valuable source of information for retail stores. This information can help predict customer behavior, evaluate marketing strategies, and find areas for improvement.

Motion tracking falls into the area of *computer vision*, which is an interdisciplinary field that deals with gaining high-level understanding of image or video data and automating tasks based on visual information. Computer vision is at the intersection of image processing, artificial intelligence, physics, and software engineering.

A major part of this work is focused on Multiple Object Tracking (MOT), the task of identifying objects in a scene and following their positions on subsequent frames. The main parts of MOT are object detection and object association between frames. Object association is also called re-identification because we are trying to find already identified objects in a new frame. While this work's goal is motion tracking of people, most of the techniques can be applied to general MOT.

Artificial inteligence (AI) and Machine learning (ML) are vital components of MOT applications. The most popular models for image data processing in the past decade have been Neural Networks (NNs) which will be introduced in chapter 1.

As AI grows increasingly common and approachable, there is more focus on performance and scalability. One approach that has been rising in popularity in the last years is *edge computing*[1], a paradigm that moves computation to the edge of the network, where the data is acquired. Processing data this way can save the time and resources needed to transport the data itself, as only processed data are transferred. Specialized hardware used for this purpose is called an edge device. The use of edge devices typically means working with limited resources, which is also a topic of this work. The advantage is that

the resulting product is better suited for real-world usage.

While movement and location information is helpful, image data provide additional information that we can use. Another part of this work focuses on retrieving age and gender data for tracked people. This information can be used in the retail environment for customer analysis and better targeted marketing.

Objectives

This thesis aims to design and implement a pipeline for tracking people in front of a retail store while also obtaining age and gender information where possible. The starting point is the research of existing approaches and solutions. The next step is experimentation and analysis of data collected in the target environment. Based on this, a pipeline will be designed and implemented with emphasis on real-life usability and deployment on edge devices.

Motivation

MOT is a natural task to consider. This task has received significant attention in research and in practice. Progress in AI theory and computer hardware has allowed MOT to be achievable with lesser budget and without expensive hardware. It provides interesting and practical use for knowledge in fields of AI, statistics, and image processing.

Furthermore, this thesis is directly related to my work at the ImproLab laboratory at FIT CTU. The results of this work will be used for practical application and real-world usage in the retail environment.

Challenges

While MOT has been actively studied, the problem is not yet solved. Real environments are complex and variable. Scenes are recorded at different angles and under different lighting conditions. Human movement patterns are complex and virtually unpredictable. This means trackers have to work with uncertain and imprecise information. Both the problems and their solutions, are explored more in-depth in the following chapters.

AI models often require large datasets for training. These datasets are also needed to tune the whole MOT algorithm and evaluate it. This presents a challenge of obtaining a representative and sufficiently large dataset. This task is currently further complicated by the specific situation related to the Covid-19 epidemic. Datasets are discussed in more detail in later parts of the work.

Assumptions

MOT is a broad topic with many possible approaches. To keep the scope manageable, this work assumes a single static camera watching a known scene. Furthermore, we are interested in solutions that work in real-time or near real-time applications on edge devices. For the task of demographic characteristics, we assume the majority of people are not wearing face masks.

Thesis structure

The rest of the thesis is organized into several chapters. Chapter 1 introduces theoretical concepts needed for understanding this work. Chapter 2 describes work related to the MOT and re-identification (ReID) tasks. Chapter 3 discusses the work's objectives in more detail and describes the dataset collection. Chapter 4 presents application design and implementation. Chapter 5 evaluates the application's results on the collected dataset, compares different detection models and benchmarks the optimization framework.

Chapter 1

Theoretical Background

This chapter introduces concepts and terms used throughout the work. It starts with a general discussion of AI and common terms used in this field. Second part discusses NNs as its the main ML model used in this theses. The next section describes Convolutional Neural Network (CNN), a special type of NN widely used in image processing. The last part introduces the Kalman filter.

1.1 Artificial Intelligence

There are many definitions of AI. [2] define AI as the study of agents that receive percepts from the environment and perform actions. Each such agent implements a function that maps percept sequences to actions. Other possibilities are to define AI as the study of either intelligent or human-like systems.

Another term associated with AI is ML, an area of AI that focuses on automatic learning of correct actions based on data. Another way to look at this is that the system autonomously gains knowledge from training data. There are two main approaches in ML - *supervised learning* and *unsupervised learning*.

In unsupervised learning, the model is trying to gain information from the dataset without explicit correct answers. The absence of correct answers leads to difficulties when evaluating the results but further reduces the need for human input. Typical tasks in unsupervised learning are based on clustering.

Supervised learning uses datasets with correctly labeled data. The availability of labels leads to a straightforward approach where the model can optimize some function related to how much its output matches the labels. The optimized function is often called *loss function*, *objective function* or *cost functions*.

1.1.1 Supervised Learning

This part introduces common concepts and approaches in supervised learning.

Main tasks of supervised learning are *classification* and *regression*. Both deal with assigning a value to some input vector. In classification, the task is to assign a label from a finite and typically small number of choices called classes. In regression, the number of possible answers is infinite, or it is practical to state the problem as if there was.

The typical supervised learning process splits the dataset into three parts. The first part is called *training* data and is used to train the model. Second part is *evaluation* data. The evaluation data is used to evaluate the performance of a trained model. The main goal of this evaluation is to find *hyperparameters*. Hyperparameters are parameters that the model does not learn on its own during training. The last part of the dataset is called *testing* data. It is used in the final stage to evaluate the model on data it has not seen yet. This evaluation enables reasonably predicting the model's performance on future data, assuming that the testing dataset is representative.

Alternative method for finding hyperparameters is *cross-validation*. Instead of splitting the dataset into fixed training, evaluation, and testing parts, the data is split only into training and testing data. Training data is then split into n parts. In each training step, we train the model n times on the training data without one part (in a way to leave out each part once). This approach can lead to more robust models and is especially useful when the dataset is relatively small. On the other hand, this increases the computing time significantly.

1.2 Neural Networks

Artificial Neural Network is a model that is used throughout this work. NNs have proved to be very useful, especially in the area of image processing. There are many types of NNs, and their use is very versatile. This section provides a basic introduction to NNs.

A basic part of NN is a neuron. An artificial neuron is a model that is inspired by a biological neuron. However, while the workings of a biological neuron are complicated, the artificial neuron is very simple. The main idea is that many simple units linked together can add up to an intelligent whole.

1.2.1 Artificial Neuron

Output of a single neuron is calculated as some function, called *activation function* applied to a weighted sum of inputs as shown in Equation 1.1, where x_i is the i-th input, w_i its weight, b is the bias, σ is the activation function

and n is the number of inputs.

$$y = \sigma\left(\sum_{i=1}^{n}(w_i x_i) + b\right) \tag{1.1}$$

1.2.2 Activation Functions

The activation function should be nonlinear. Linear functions are not useful here because the composition of linear functions is a linear function, so we could easily replace multiple neurons with one neuron with different weights. Non-linearity is also needed to fit nonlinear data.

Activation functions are usually required to be differentiable. The differentiability is needed for *backpropagation* algorithm, which is an algorithm for efficient training of NNs that will be introduced later.

Common activation functions are:

- sigmoid: $\sigma(x) = \frac{1}{1+e^{-x}}$,
- hyperbolic tangent: $\tanh(x) = \frac{e^{2x}-1}{e^{2x}+1}$,
- ReLU: $\mathrm{ReLU}(x) = \max(0, x)$.

The last layer typically uses different activation functions based on the target task. For binary classification the typical function is *logistic sigmoid*

$$f(\xi) = \frac{1}{1+e^{-\xi}} = \frac{e^{\xi}}{1+e^{\xi}}$$

and the resulting value is interpreted as the probability that the given input is from class 1. This can be written as $\hat{\mathrm{P}}(Y = 1|X = x)$.

For classification into c classes a *softmax* function is used with c output neurons. Output for i-th neuron is

$$f_i(\xi) = \frac{e^{\xi_i}}{e^{\xi_1} + \ldots + e^{\xi_c}},$$

where $\xi = (\xi_1, \ldots, \xi_c)^T$ and ξ_i is the input for i-th output neuron. The interpretation is similar as for the binary case, formally $f_i(\xi) = \hat{\mathrm{P}}(Y = i|X = x)$. Final prediction is then the class with maximum probability assigned

$$\hat{Y} = \underset{i \in 1,\ldots,c}{\operatorname{argmax}} f_i(\xi).$$

1.2.3 Feed Forward Neural Network

A feedforward neural network is a basic type of NN with neurons organized into layers. The first layer is called *input layer* and represents input variables. Last layer is called *output layer*. The remaining layers are called *hidden layers*.

NNs with large number of hidden layers are sometimes called *deep neural networks*. Usage and study of such NNs is sometimes called deep learning. There is however no consensus on the precise meaning of the term. In practice, the term deep learning is often synonymous with learning of NNs.

Every neuron in each layer is connected to neurons in the following layers creating a directed acyclic graph.

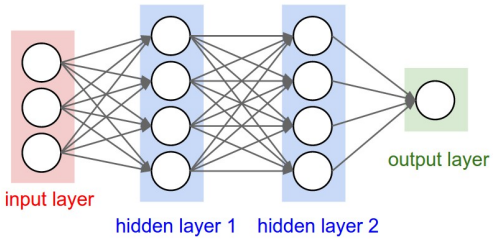

Figure 1.1: Structure of a simple neural network.[3]

Let $w^l_{i,j}$ be the weight of connection from i-th neuron in $(l-1)$-th layer to j-th neuron in l-th layer. Let b^l_j be the bias of j-th neuron in l-th layer, σ some activation function and $N^{(l)}$ number of neurons in layer l. Then the activation (output) a^l_j for the j-th neuron in l-th layer is

$$a^l_j = \sigma\left(\sum_{i=1}^{N^{(l-1)}} (w^l_{i,j} a^{l-1}_i) + b^l_j\right). \tag{1.2}$$

We can write this more succinctly with the usage of matrices and vectors. Let W^l be a weight matrix for layer l which has $w^l_{i,j}$ from Equation 1.2 in j-th row and i-th column. Similarly let $b^l = (b^l_1, \ldots, b^l_{N^{(l)}})$ be a bias vector. Then we can compute an activation vector a^l whose components are activations a^l_j with

$$a^l = \sigma\left(W^l a^{l-1} + b^l\right) \tag{1.3}$$

Each layer produces a nonlinear transformation of outputs from previous layers. [4] proved that standard multilayer feedforward networks with as few as one hidden layer are capable of approximating any Borel measurable function from one finite-dimensional space to another to any desired degree of accuracy. In this sense, multilayer feedforward networks are universal approximators. However, finding parameters for such networks is rather difficult.

[5] argues that shallow architectures can be very inefficient in terms of the required number of computational elements and examples. Furthermore, they argue that deep architectures have the potential to generalize in a way that is crucial to make progress on the kind of complex tasks required for artificial intelligence. This corresponds well with empiric observations. Commonly used NNs have tens of layers and millions of parameters [6, 7, 8].

1.2.4 Learning

The goal of learning is to find the parameters $\theta = (W, b)$ (from Equation 1.3) that minimize the selected loss function $L(\theta)$. Common loss functions are categorical cross-entropy for multi-class classification and Mean Squared Error (MSE) for regression.

Let θ be learned parameters, Y the vector of target labels, also called *ground truth*, and $\hat{Y}$ vector of predicted values from NN based on θ and N input vectors. Let $||.||$ be L2 norm. Then MSE is defined as:

$$L(\theta) = \frac{1}{N} \left|\left| Y - \hat{Y} \right|\right|. \tag{1.4}$$

With the use of a suitable cost function such as MSE and differentiable activation functions the whole NN is differentiable and can be trained using *backpropagation*.

Backpropagation is an iterative algorithm, where we compute the gradient of the cost function with respect to the weight and then update the weights with a step proportional to the negative of the gradient. The algorithm is explained in more detail, for example in [9].

1.3 Convolutional Neural Networks

This section introduces CNNs, which are a specialized kind of NN for processing data with a known grid-like structure, for example, time-series data or images. Convolutional networks have been very successful in practical applications. The information in this section is mainly based on [10] and [3].

CNN is a NN that uses *convolution* instead of general matrix multiplication in at least one of its layers. Convolution is a mathematical operation defined for functions f and g as

$$(f * g)(t) = \int_{-\infty}^{\infty} f(x)\, g(t - x)\, dx$$

in continuous and

$$(f * g)(t) = \sum_{a=-\infty}^{\infty} f(a)\, g(t - a)$$

in the discrete case. The first argument f is called *input* and the second argument g is called *kernel* of the convolution. The output is sometimes referred to as the *feature map*. Convolution can be generalized to multiple dimensions.

In machine learning applications, the input is usually a *tensor* (multidimensional array) of data, and the kernel is usually a tensor of learned parameters. In practice, both input and kernel are considered zero everywhere but

in the finite set of points where we store their values. This allows the implementation of the infinite summation as a summation over a finite number of elements.

In traditional neural networks, the neurons in each layer are connected to all neurons from the previous layer. This large number of connections leads to a large number of parameters that need to be learned. CNNs leverage the structure of input data to reduce the number of parameters. If we assume that the input consists of images, we can reasonably constrain the neural network architecture.

The layers of CNNs are arranged in three dimensions: width, height, and depth (which refers to the third dimension of an activation volume). The input layer holds the image, so its width and height will be the dimensions of the image, and the depth would be three for color images (representing the three Red, Green, Blue (RGB) channels).

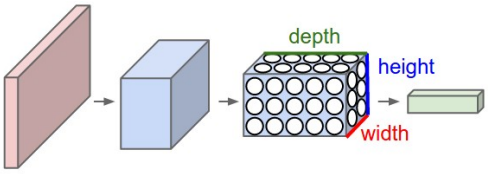

Figure 1.2: CNN layer transformation of 3D input volume to 3D output volume.[3]

Three main types of layers are used to build CNNs: *convolutional layer*, *pooling layer* and *fully-connected layer*. Fully-connected layers were already introduced in section 1.2.3.

1.3.1 Convolutional Layer

A convolutional layer is the core building block of CNN. Each layer consists of a set of learnable *filters*. Every filter is spatially small and has the depth of the input volume. Each filter is convolved across the input volume's width and height to produce a two-dimensional activation map. Intuitively, the network will learn filters that activate when they see some feature. The idea is that early layers learn to recognize simple features like an edge, and later layers will learn to recognize more complicated patterns based on these features. The output of the whole convolutional layer is the output of each filter in given layers stacked along the depth dimension - this produces the output volume.

Every entry in the 3D output volume can be interpreted as an output of a neuron looking at only a small region in the input and sharing parameters with all the neurons that apply the same filter. This property is called *local connectivity*.

The spatial extent of this connectivity is determined by a hyperparameter called *filter size*. The filter size only affects the spatial dimensions (width and

height). The connectivity along the depth axis is always equal to the depth of the input volume.

Convolutional layers use a schema called *parameter sharing* to reduce the number of parameters greatly. This reduction is based on the assumption that if one feature is useful to compute at some spacial position (x, y), it should also be useful to compute at different positions. We can then constrain the neurons in each depth slice to use the same weights and bias. It is common to refer to this shared set of weights as a *filter* (or a *kernel*).

1.3.2 Pooling Layer

Pooling layers perform non-linear downsampling of the input. They do this by combining multiple values into a single value that they pass to the next layer. Most common approach is *max pooling*. Max pooling takes the maximum value from its input.

It is common to periodically insert a pooling layer between convolutional layers in a CNN architecture. The layer's function is to reduce the number of parameters, which reduces the number of computations needed and helps control overfitting.

1.3.3 Transfer Learning

Transfer learning is a standard process, where we take a pre-trained model for a similar task and use it as initialization or a feature extractor for the target task. The use of pre-trained models dramatically reduces the computational power needed and the need for an extensive dataset.

The use of an existing CNN as a feature extractor is simple, as only the last fully-connected layer needs to be removed or replaced.

1.4 Kalman Filter

This section introduces *Kalman filter*, which is an integral part of tracking algorithms used in this work. The information in this section is mainly based on [11].

Kalman filter is a mathematical model to gain (relatively) precise information about a system based on imprecise measurements and information about the system. Kalman filters are fairly general and have usages in estimation, data smoothing, and control applications. We will focus mainly on its usage in tracking applications.

1.4.1 Introduction to g-h Filters

Any real measurement is inaccurate. The output of any sensor does not give us perfect information about the observed system but depends on the sensor's

quality. To deal with this, we can use an algorithm called *g-h filter* (also called alpha-beta filter).

The first idea of the g-h filter is that the system's behavior should influence how we interpret the measurements. Imagine we are weighting a rock and getting slightly different results each time. We would probably attribute these differences to noise in the measurement. On the other hand, if we were getting changing position from a car GPS, we might conclude that the car is moving.

Assume we have some predictions for the target variable. If we only form estimates from the measurements, then the prediction will not affect the result. If we only form estimates from the prediction, then the measurements will be ignored. This leads to the second idea that we need to take some combination of the prediction and measurement. We will call the difference between the measurement and prediction the *residual*.

In general, we cannot expect to know the rate of change of the target variable, and it also may change over time. These ideas lead to an iterative two-step process. First, we *predict* the target variable and its rate of change. Next, we *update* the target variable and its rate of change based on the prediction and new measurement.

This algorithm is very general. Kalman filter is then one approach on how to do these steps.

1.4.2 Kalman Filter Algorithm

Like any g-h filter, the Kalman filter makes a prediction, reads a measurement, and then forms a new estimate between the two.

The Kalman filter is using normal distributions for the representation of measurements and predictions. The normal distribution is well studied and has many interesting properties. Using normals allows us to store information about whole probability distribution as just two numbers - mean μ and variance σ^2.

Sum of two normal distributions $N(\mu_1, \sigma_1^2), N(\mu_2, \sigma_2^2)$ is a normal distribution $N(\mu_1 + \mu_2, \sigma_1^2 + \sigma_2^2)$. The product of two normal distributions is proportional to a normal distribution, meaning we can scale it to a normal distribution. These two properties mean we can sum and multiply normal distributions, and the result will still be a normal distribution (assuming we are normalizing after multiplication).

Predict

The general formula for the predicting the next state mean is

$$\overline{x} = Fx + Bu. \tag{1.5}$$

x denotes the state mean. F is the *state transition function*. B and u let us model control inputs to the system and can be removed if we do not have any control over it.

State covariance $\overline{P}$ is predicted with

$$\overline{P} = FPF^T + Q, \tag{1.6}$$

where P is the previosu state covariance, F is the state transition function from Equation 1.5 and Q is the process covariance.

Update

The update step consists of applying the following equations.

$$y = z - H\overline{x} \tag{1.7}$$
$$K = \overline{P}H^T(H\overline{P}H^T + R)^{-1} \tag{1.8}$$
$$x = \overline{x} + Ky \tag{1.9}$$
$$P = (I - KH)\overline{P} \tag{1.10}$$

$\overline{x}, F$ and $\overline{P}, Q$ are from equations 1.5 and 1.6 respectively. H is the measurement function. z and R are the measurement mean and *noise covariance*. K is called *Kalman gain*. I is the identity matrix.

Measurement noise is the variance of the sensor we are using, while process noise is the observed system variance. The measurement function maps the true state space into the observed space. The Kalman gain is the relative weight given to the measurements and current state estimate. With a high gain, the filter places more weight on the most recent measurements.

Summary

Kalman filter is a recursive algorithm that can be used to extract useful information from noisy measurements.

In the context of tracking, the Kalman filter can be used to better approximate track's bounding boxes. The Kalman state is typically some representation of a rectangle and its speed. The measurements are usually taken from a CNN. These measurements are noisy, and the Kalman filter smooths them to provide a more accurate and stable position. Furthermore, we can also predict the track's position in the next frame, which is used to match the track to new detections.

The filter needs correctly designed models and functions introduced in previous sections to work correctly. There is no universal approach, and the design must be based on experience, intuition, and experimentation. One possible design is described in section 4.2.

Chapter 2

Related Works

This chapter presents relevant work in the area of MOT and age and gender recognition.

2.1 Multiple Object Tracking

Multiple Object Tracking is a longstanding goal in computer vision[12, 13, 14], which aims to estimate trajectories for objects of interest in videos.

Tracking-by-detection has emerged as the preferred paradigm to solve the MOT problem[14, 15]. This paradigm simplifies the task by breaking it into two steps: detecting the objects' locations independently in each frame and then forming tracks by associating corresponding detections across time. The second step is sometimes called linking or ReID.

In recent years, NN based detectors have clearly outperformed all other methods for detection.[16, 8].

Track association has been handled by various methods. Straightforward Intersection over Union (IOU)[1] based approach has been applied[17] as well as various embeddings from NNs[15]. The association step usually first computes a cost matrix based on the motion and appearance information and then matches the tracks to minimize the total cost.

When using the two-step method, one can develop the most suitable model for both tasks separately. Additionally, one can crop and resize the image patches based on the bounding boxes before estimating the ReID features.

Recently [12] came up with a model that handles both the detection and ReID tasks while achieving accuracy comparable to state-of-the-art (SOTA) trackers.[14]

An alternative approach using recurrent neural networks for data association has been explored in [18] and [19]. While providing some advantages, their work is not competitive with current SOTA methods.[14]

[1] IOU of two areas is the area of their overlap over the area of their union.

2.1.1 Simple Online Realtime Tracking

Simple Online Realtime Tracking (SORT) is a pragmatic approach to MOT with a focus on simplicity and performance introduced in [13], which uses Kalman Filter (introduced in section 1.4) to predict object location in the next frame. Cost matrix is based on IOU of Kalman predictions and detections in the new frame. Finally, Hungarian algorithm[20] is adopted to make a minimum cost matching based on the IOU.

The main disadvantage of the SORT algorithm is its reliance only on position and movement data. This can easily lead to identity switches of tracks when occluded either by environment or by other tracks.

Simple Online and Realtime Tracking with a Deep Association Metric (DeepSORT) extends the SORT with appearance information from a CNN.

To incorporate motion information DeepSORT uses Mahalanobis distance between predicted Kalman states and newly arrived measurement:

$$d^{(1)}(i,j) = (d_j - y_i)^T S_i^{-1}(d_j - y_i),$$

where (y_i, S_i) is the projection of the i-th track into measurement space and d_j is the j-th bounding box detection. The Mahalanobis distance takes state estimation uncertainty into account by measuring how many standard deviations the detection is away from the mean track location. This metric makes it possible to exclude unlikely associations by thresholding the Mahalanobis distance. The threshold is calculated as a 95% confidence interval computed from the inverse χ^2 distribution.

To incorporate appearance information we compute an appearance descriptor r_j for each detection d_j with $||r_j|| = 1$. Furthermore, we keep a history $\mathcal{R}_k$ of the last L_k descriptors for each track k. We then measure the distance between the i-th track and j-th detection as the smallest cosine distance:

$$d^{(2)}(i,j) = \min\{1 - r_j^T r_k^{(i)} \mid r_k^{(i)} \in \mathcal{R}_i\}.$$

We can also find a suitable threshold to indicate if an association is admissible according to this metric using a training dataset.

We can combine both motion-based information from Mahalanobis distance and appearance-based information from the cosine distance using a weighted sum

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

where we call an association admissible if it is admissible for both thresholds described above.

The influence of each metric can be controlled through the hyperparameter λ.

2.1.2 Metrics

To evaluate and compare different methods, we need a way to measure errors. While this is very straightforward for some tasks, this is not the case for MOT. [21] introduces two relatively simple and intuitive metrics that will be described in this section. Both metrics are widely used[14].

The first metric is called Multiple Object Tracking Precision (MOTP) and characterizes trackers precision in estimating object positions. The second metric is Multiple Object Tracking Accuracy (MOTA) and expresses the tracker's ability to determine correct object configuration and keep consistent tracks.

The procedure for calculating these metrics consists of three steps each frame:

1. establish the best possible correspondence between hypotheses and objects,

2. for each correspondence compute the error in objects position estimation,

3. accumulate following errors:

   - count all objects with no hypothesis as misses (*false negatives*),
   - count all hypotheses with no real objects associated as *false positives*,
   - count all occurrences where the tracking hypothesis for an object changed compared to previous frames as *mismatches*.

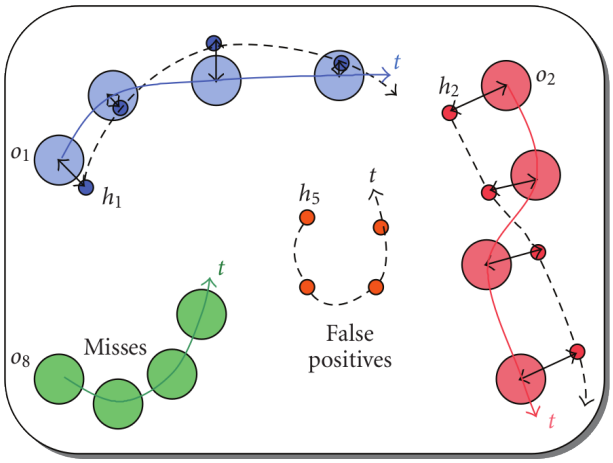

Figure 2.1: Illustration of various types of errors.[21]

Let c_t be the number of matches for time t. For each match, let d_t^i be the distance between the object and the hypothesis. The MOTP is then defined as:

$$\text{MOTP} = \frac{\sum_{i,t} d_t^i}{\sum_t c_t}.$$

Let m_t be the number of misses, fp_t the number of false positives, mme_t the number of mismatches and g_t total number of objects in time t. The MOTA is then defined as:

$$\text{MOTA} = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}.$$

The MOTA can be seen as computed from three ratios - miss ratio, false positives ratio, and mismatch ratio.

For more discussion and implementation details see [21].

2.2 Person Re-identification

Person ReID is a fundamental task for people MOT. One person's appearance can change significantly in different frames, for example, by changing pose, turning around, or taking off a backpack. On the other hand, people often wear similar clothes and may look very similar, especially when viewed from a distance. These variations make the task challenging.

[7] presents *OSNet*, a CNN architecture for tackling the ReID task. While CNNs have been used before (for example in [15]) to learn discriminative features for ReID, *OSnet* presents a novel approach.

Key concept in *OSnet* is focus on *omni-scale* feature learning and its effective implementation. Authors argue that using even features at multiple scales (for example, local and global features) is not sufficient and features of all scales are crucial for the ReID task.

The result is a lightweight ReID network that achieves SOTA results on multiple datasets outperforming even much bigger models.[7]

Chapter 3

Analysis

The main goal of this work is to create a pipeline for processing video data, with the goal of consistently tracking people in front of a retail shop. Additionally, we want to extract age and gender information for found tracks.

This chapter discusses this objective in more detail to allow us to design and evaluate a solution. To keep the scope manageable while keeping the application usable in a real-world environment, we have to make some assumptions about the observed environment (inputs). These assumptions should be noted so the limitations of the system are clear.

3.1 Target Environment

The target environment is an area in front of a retail shop. This area can be outdoors or indoors, for example, inside a shopping mall.

We assume a single stationary camera recording this environment. Each environment is different, so the setup must be adjusted individually to provide the best possible video quality. A specific setup used for data acquisition for this work will be described in a later chapter.

Since only one camera can be used, it must be carefully positioned to capture the whole area of interest with reasonable quality. The area is also expected to be well lit, meaning the system is not expected to work, for example, at night, unless suitable artificial lighting is provided.

On the other hand, imperfect conditions are expected in real environments. The system should deal with minor lighting changes and reflections caused by the environment and various distortions caused by the camera. For example, reflections from the shop windows are expected. The camera system should be selected and installed in a way to minimize these problems.

3.2 Dataset

An appropriate dataset is required to tune and evaluate the algorithm. [14] presents multiple datasets from various scenes along with annotations. These datasets are commonly used for evaluation in literature. Both datasets and evaluation results are available at `https://motchallenge.net`. This dataset's main advantages are that it allows for direct comparison with many different tracking algorithms and provides ground truth annotations.

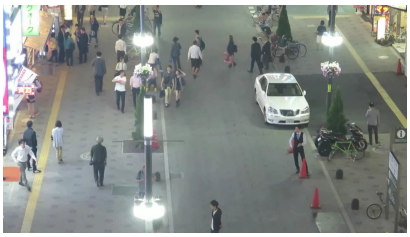

Figure 3.1: Example frame from the [14] dataset.

We have decided to create our dataset targeting the retail environment, as we have not found any usable data from the specified environment. Such a dataset will be more representative and allow for more accurate evaluation. Further, it can be used to optimize and fine-tune the system for the target environment.

Dataset collection was done in cooperation with store owners, where the designed system might be used in the future. This cooperation allowed us to collect the dataset according to the system's assumed use. Collecting the dataset in the retail environment has shown some complications the system might face in actual usage and helped significantly with problem analysis from a practical standpoint.

The dataset collection process was done across two locations. The first location was used for selecting a camera, finding suitable camera placement, and initial experiments. The dataset itself was collected at the second location.

3.2.1 Camera Selection

This section describes the first part of the dataset collection process, where short videos were recorded with multiple cameras in different positions at the first location.

Cameras were placed in a shop window behind glass with the view facing the street. Evaluation criteria were image quality, camera view (does the camera see the full Region of Interest (ROI)), and camera noticeability. Camera noticeability is meant as a criterium of how much the camera is visible to a passerby, as a noticeable camera might discourage potential customers from browsing the shop window.

Three possible camera placement configurations were considered:

1. at the edge of the shop window, near the glass, at approximately 150 cm from the ground,

2. at the center of the shop window, near the glass, at approximately 150 cm from the ground,

3. at the edge of the window in the corner, at approximately 220 cm from the ground, positioned at an angle.

The first option did not present a sufficient view of the ROI and was rejected. The second option provided good image quality while being more noticeable. The third option proved to be very unobtrusive with a good view. However, the image quality seemed subjectively slightly lower, mainly thanks to reflections on the glass window.

Recordings from the second and third configurations were further evaluated using a simple initial version of the tracking algorithm. This early evaluation confirmed the third configuration as suitable and hinted at the task as being reasonably solvable.

Based on the initial testing, the AXIS FA1105 surveillance camera[22] was selected for the following recordings. This camera is highly discreet, provides sufficient video quality with resolution 1920x1080 (1080p), and has a wide 111° horizontal field of view.

3.2.2 Dataset Acquisition

Before starting the dataset collection itself, we needed to find a suitable camera configuration for the second location, which proved to be more challenging than expected. The camera was placed at a shop inside a shopping mall. The main difficulties were caused by camera obtrusiveness and appearance, lighting conditions, and reflections.

The camera appearance issue was solved by 3D printing a custom camera holder, which allowed for a more discrete and pleasant camera look. One of the main lighting problems was direct lighting from the shopping mall ceiling, which was handled by adding a black cover on top of the camera to shield it from this lighting. The camera is shown in figure 3.2.

Another significant problem was reflections on the shops' glass windows. A polarization filter was added to the camera to minimize these reflections. While this improved the image quality, reflections remain a problem. The effect can be seen in figure 3.3.

The camera remained in the location long-term, however usable dataset size is limited by the time needed to annotate the data. Multiple video sequences were hand-selected and annotated using CVAT software[23]. The total dataset size is 2600 annotated frames. A sample dataset frame can been seen in figure 3.4.

Figure 3.2: Camera used for dataset acquisition.

Figure 3.3: Image taken without a polarizer filter (left) and with polarizer filter (right).

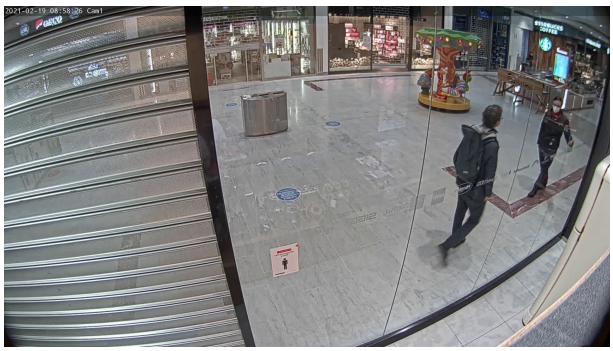

Figure 3.4: Sample frame from collected dataset.

3.2.3 Region of Interest

The goal of our work is to observe a region in front of a shop. It can be expected that the camera captures a larger area, as is the case in our collected dataset. The tracks need to be filtered based on their position to select only the tracks in the target area to provide relevant statistics.

The ROI is also relevant for the experimental evaluation. Evaluating tracks only in ROI makes the evaluation more relevant to the actual goal. Tracking people far away from the shop (and the camera) is not our goal and may not be reasonably achievable. Tracks in a significant distance are small, their image resolution is low, and occlusions and bounding box overlaps make this

even more difficult. What is considered relevant needs to be considered for each camera setup individually.

To filter the relevant tracks, we need to specify a function to tell if a given track lies inside the ROI. The target area could be intuitively specified as a polygon. More general shapes could allow more flexibility but increase the complexity of operations such as intersection. Once we have the target area specified as some geometric shape, we can find if a track is inside based on bounding box intersection. Simple intersection could also be expanded to consider, for example, only tracks above some intersection over minimum threshold. Another possible approach is to convert each track to a single point (such as its bounding box center) and then find if the given point lies in the ROI.

The methodology used for evaluation is described in chapter 5.

3.2.4 Age and Gender Information

The original goal for the dataset was to include age and gender information. This was an additional reason for collecting our dataset, as we do not know any MOT dataset that includes the biometric information. The current Covid-19 epidemic complicates the task significantly, as (nearly) all people wear face masks.

Initial experiments on collected data confirmed that extracting biometric information on images with face masks is challenging and currently available models and datasets are not sufficient for this task. Furthermore, we did not find any relevant datasets and little relevant work, which is probably caused by how unexpected and novel the current pandemic situation is.

Dealing with face masks properly is out of scope for this work. We consider the mask situation temporary, so it is not essential for future use of the application.

Based on the current difficult situation, we have made the following decisions. We do not include the age and gender information in our collected dataset. We include the age and gender classification in our pipeline; it is prepared for use once the situation with face masks changes. We evaluate the age and gender models mainly from the performance standpoint.

3.3 Age and Gender Classification

One of the goals of this thesis is to extract age and gender information for tracked people. Multiple works dealing with this task exist, using various CNN architectures[24, 25, 26].

The same network architecture can typically be used for both age and gender as in [24], except the last layer, which has to match the target number of classes.

The expected output for gender is either male or female. For age information, the output format is less straightforward.

While age estimation has been formulated as a regression problem, for example, in [25][2], it is more common to formulate it as a classification problem, where the categories are various age ranges[24, 26].

Formulating the age prediction as a classification task into some age ranges simplifies the task (as we do not try to predict the exact age) and arguably does not reduce the information usefulness noticeably. Marketing strategies and behavior prediction will probably be different for various age groups, such as children, adults, and the elderly, but differ less inside these groups.

3.3.1 Classification Input

The required information can be extracted either from a whole-body image or from a face image. Using a whole-body image would be very beneficial since this information is always available, and no association step is needed. [27] explores gender classification based on body pose estimation, which is in turn based on image information. We experimented briefly with this approach and found both performance and accuracy to be insufficient.

A more typical approach is to use face information[24, 25, 26], which has its own downsides. The face may not always be visible, and we need to associate faces to appropriate tracks.

In contrast to classification on a single image, our input is a sequence of frames. We need for a given face to be visible on at least one frame to make predictions. While this provides no guarantees, it makes the chance of a successful face detection more likely. If we have multiple predictions for a single track, we need to put them together using some statistical function such as mean or median.

3.3.2 Face Alignment

Both literature[26] and our experiments suggest that the classification task is heavily influenced by face alignment. We found that many detected faces are practically unusable for prediction because of alignment and general image quality issues.

As a potential improvement, we experiment with filtering faces based on face alignment. The goal is to accept predictions that are based only on face images with reasonable quality and alignment.

3.4 Hardware and Performance

Most tracking and age and gender classification methods use NNs as described in previous chapters. One of the limiting factors of NNs is the computing

[2]Even [25] is based on classification that is turned into regression using expected values.

power they require[14, 8]. Advances both in theoretical understanding and hardware have allowed for NNs to be used in an increasing number of devices such as mobile phones[6] and even browsers[28].

However, video processing is still a very data-intensive task. Processing live feed requires processing multiple images each second. One of the system's primary goals should be to focus on speed to allow live camera feed processing.

Furthermore, using a dedicated (edge) device that would process the video stream at the camera's location would significantly improve the system's scalability and ease of use.

For these reasons, Xavier Jetson NX (Jetson)[29] was chosen as a testing device, which will be used to process the video stream and run the tracking algorithm in our experiments. This device is very compact and specialized for both video processing and NNs inference, making it suitable for use in the retail environment. Running all experiments on single hardware assures that the results are comparable when looking at processing time. Running on a suitable device for the production environment also makes the results more directly relevant and usable.

3.4.1 Optimization

In recent years, there has been growing interest in building AI models with the focus not only on quality but also on performance (computation power required)[30, 31, 8]. Performance can often be significantly increased when using a smaller model[3] without significant quality loss[25].

Another important topic is optimization of existing models. The task of reducing NN size by removing parameters is called *pruning*[32].

A common pruning strategy[32] is to first train the target NN to convergence. After which parameters or structural elements are issued a score. The network is then pruned based on these scores. Pruning typically reduces[4] the accuracy of the network, so the network can then be trained further (this is called *fine-tuning*).

NVIDIA TensorRT[34] is a framework for NN optimization and efficient inference. This software is closed-source, and the precise optimization algorithm is not disclosed. We will evaluate it experimentally in chapter 5.

[3] By smaller, we mean model with less learnable parameters. For NNs the critical factor is typically depth; however, the overall architecture is also important.

[4] Pruning can also increase the accuracy in some cases[33].

Chapter 4

Design

This chapter presents our proposed tracking algorithm. Our solution is based on the DeepSORT[15] algorithm, introduced in section 2.1.1. Implementation, especially with respect to efficient use of hardware, has been inspired by [30].

We use the following notation. *Track* is each unique object of interest (in our case person). Track's *age* is the number of frames it has not been associated with any detection. We will say a track is *active* if its age is below some threshold. We say track is *confirmed* if it has been associated with a detection at least n times. Track is considered *lost* if it moves out of the frame or is not matched with a detection for m frames.

The algorithm consists of the following high-level steps which are run for each input frame.

1. detect people and faces
2. extract visual features from detections
3. apply Kalman filter
   a) run prediction for each existing track
   b) mark/remove tracks that move out of frame
4. associate existing tracks with detections and update tracks
   a) associate *confirmed* tracks based on Mahalanobis distance and visual features
   b) associate remaining *confirmed* and *active* tracks based on IOU
   c) associate *unconfirmed* tracks based on IOU
   d) associate (ReID) *lost* tracks based on visual features
   e) update tracks
   f) register new tracks

5. extract biometric information from faces

6. associate faces to tracks

(a) Tracks at time t.

(b) Detections at $t + 1$.

(c) Kalman predictions for $t + 1$.

(d) Matching is based on IOU and visual features.

Figure 4.1: Visualisation of the main tracking steps. Images from [14], modified.

4.1 Detection and Feature Extraction

CNNs, which were introduced in section 1.3, provide SOTA results for the tasks of detection, feature extraction and age and gender classification. Many CNN architectures exist, and the choice of the appropriate one is not obvious. The training dataset selection is also essential.

Our criteria for model selection are accuracy, speed, and, for practical reasons, availability of pre-trained models.

Since performance is a priority, the model should detect both faces and people. Furthermore, we could increase performance if the network used for detection also provided visual features, which would remove the need for a dedicated feature extractor network. [12] proposes such a network while claiming good performance. We found their model to be too slow for real-time processing on our hardware. However, their approach seems very promising, and combining detection and feature extraction might be the best approach in the future.

Comparison of various models is presented in chapter 5.

Based on our analysis, we selected YOLOv4[35] model as the detection model. We use version pre-trained on [36] dataset, with potential fine-tuning on data from the target environment.

For feature extraction we use *OSnet* architecture from [7], which achieves SOTA results on multiple ReID datasets and is very lightweight. For details see section 2.2. Specifically we use the the version termed `osnet_x0_25` trained on the MSMT17[37] dataset. Pre-trained model is provided by the paper authors.

4.2 Kalman Filter Design

We use the Kalman filter to predict each track's position and update its position after association with a detection. General introduction to Kalman filter is presented in 1.4. This section describes the Kalman model used and its application in our algorithm.

4.2.1 Model Design

We define the *Kalman state* as a vector

$$x = (x_1, y_1, x_2, y_2, \dot{x}_1, \dot{y}_1, \dot{x}_2, \dot{y}_2),$$

where the first four elements represent the coordinates of the top left and bottom right points of the track's bounding box, and the remaining elements are their respective velocities.

Each track is then represented by state means vector $x \in \mathbb{R}^8$ and covariance matrix $P \in \mathbb{R}^{8,8}$. The means vector is initialized from detection with its coordinates and zero velocity. We initialize the covariance matrix as a diagonal matrix. The specific values depend on the observed scene and quality of the detector model.

We assume a constant velocity model for the tracked objects. This assumption is common in literature [13, 15, 11]. Human motion is generally not linear. However, the Kalman filter can reasonably work even when the assumption is not satisfied.

With the constant velocity model in mind, we can define *state transition function* F as

$$F = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

Next, we define *measurement function* H, which is used to transition from the Kalman state space to a measurement space. In our case, this means moving from an 8-dimensional vector with position and velocity to a 4-dimensional

vector with only the position. The measurement function is

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

The remaining parts to define are *measurement noise* matrix R and *process noise* matrix Q. We define the measurement matrix as a diagonal matrix $R \in \mathbb{R}^{4,4}$ with $\alpha \in \mathbb{R}^+$ on the diagonal. In practice the α value is a hyperpameter based on the precision of the underlying detector model.

We model the process noise as a discrete white noise. Let $\beta \in \mathbb{R}^+$ be a hyperparameter, then the process noise matrix is

$$Q = \beta \cdot \begin{bmatrix} 0.25 & 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0.25 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 0.25 & 0 & 0 & 0 & 0.5 & 0 \\ 0 & 0 & 0 & 0.25 & 0 & 0 & 0 & 0.5 \\ 0.5 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0.5 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0.5 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 & 1 \end{bmatrix}.$$

4.2.2 Predict and Update

Predict and *update* are the basic steps of the Kalman algorithm. In the prediction part, we try to predict the Kalman state for the next time step.

The update step is based on a measurement z. In our algorithm, the measurement is a bounding box of detection associated with the given track. Update consists of computing *residual* y and *Kalman gain* K and then updating the Kalman state. Kalman gain affects how much weight we place on the measurement when combining it with the prediction.

Let $x_t \in \mathbb{R}^8, P_t \in \mathbb{R}^{8,8}$ be the state mean and covariance of the given track at time step t. State mean and covariance are separate for each track and time step.

Further, let F, H, Q, R be the various matrices defined in the previous section.

The *predict* step is described by the following equations:

$$\hat{x}_{t+1} = Fx_t, \tag{4.1}$$

$$\hat{P}_{t+1} = FPF^T + Q, \tag{4.2}$$

and the standard update step is described by:

$$y = z - H\hat{x}_{t+1}, \tag{4.3}$$

$$K = \hat{P}_{t+1}H^T(H\hat{P}_{t+1}H^T + R)^{-1}, \tag{4.4}$$

$$x_{t+1} = \hat{x}_{t+1} + Ky, \tag{4.5}$$

$$P_{t+1} = (I - KH)\hat{P}_{t+1}. \tag{4.6}$$

We replace equation 4.6 with the following, more numerically stable version from [38]:

$$P_{t+1} = (I - KH)\hat{P}_{t+1}(I - KH)^T + KRK^T.$$

4.2.3 Application

We run the Kalman filter on each input frame. First, the *prediction step* is run, and the predicted Kalman state is assigned as a new state to all tracks. This new state is then used for all operations in the algorithm's association step (described in the next section). If the intersection over minimum of the input frame and the track's newly computed bounding box is less than 0.5, the track is considered to have left the scene and is marked as lost.

When using the Mahalanobis distance, both mean and covariance are used for the distance calculation. However, when using IOU between detections and tracks, only the bounding box information is required. The bounding box information can be extracted easily using the measurement function[5].

After the detections and tracks are associated, we run the *update step* for each track for which detection was associated. The input measurement is the bounding box of the detection. We also check if the track left the scene same way as in the prediction step.

4.3 Track Association

Associating tracks and detections is a non-trivial task. The tracker has to differentiate between missed detections and objects leaving the scene. It has to decide wherever unmatched detection is a new track or just a false positive.

To make the tracker more robust and accommodate for various errors in underlying models, we perform the matching in multiple steps. We prioritize already established tracks and wait for multiple frames before marking the track as confirmed.

We find the identity switches as the most relevant and detrimental type of error, because they introduce long-term errors, whereas the false positives and false negatives are typically short-term and do not disrupt the overall trajectory.

[5] In our case, this simply means taking the first four elements of the state, but in general, the transformation could be more complex.

Identity switches can easily happen in groups of multiple people, where the algorithm cannot rely on the position information. Moreover, even relevant visual information can be difficult to obtain due to occlusion. Problematic situations may also be caused by partial or full occlusions from the environment and different visual conditions in various parts of the input frame.

4.3.1 Association Steps

The main part of all association steps is forming a cost matrix for a subset of tracks and a subset of detections. The cost matrix contains the distance between given tracks and detections. Once we have this matrix, we can easily find the minimum cost matching using one of the standard algorithms in polynomial time[20, 39]. We start the association with all defections[6]. Detections matched in any step are not used in subsequent steps.

The first association step is analogical to DeepSORT[15], which is described in section 2.1.1, with differences described further. The remaining steps are mostly inspired by [30].

In the first step, we associate only *confirmed* tracks. The cost matrix is computed based on Mahalanobis between tracks' states and detections' bounding boxes and feature vectors similarity. Apart from using only the confirmed tracks, another difference from the original DeepSORT is how we store the visual features. The original keeps a gallery of the last n features for each track and computes the distance as a minimum distance between any feature in the gallery and the bounding box feature.

Instead of this, we keep only a single feature (vector) for each track. On update, the new feature is calculated as a weighted average of the current and new feature and is normalized afterward. This approach is faster since we need to compute only a single distance for each track. An additional advantage is a potential for robustness. When using the gallery, a single incorrect measurement can take precedence over multiple valid measurements and can remain in the gallery for a long time.

In the second step, we associate the *confirmed* tracks which were not associated with any detection in the first step. We also filter out tracks that are not active. The cost matrix is computed based only on IOU of tracks and detections' bounding boxes.

The third step associates unconfirmed detections. The cost matrix is computed from IOU of tracks and detections' bounding boxes, as in the previous step.

After that, *lost* tracks are associated based only on visual features. This step is similar to the first one; however, no position information is used, as we cannot meaningfully predict the position of lost tracks.

[6]By all detections we mean all detections of appropriate class with at least given confidence.

Next, we update all matched tracks and set their age to zero. We update their Kalman filter state as described in the previous section.

The next part is dealing with unassociated tracks. We remove those which are unconfirmed, dismissing them as a false positive. We increase the age of remaining unmatched tracks. If the track's age is above some threshold, we mark it as lost.

The last step is creating new tracks from the remaining detections. We initialize each track with a Kalman state as described in section 4.2.1.

4.4 Age and Gender Classification

This section describes the age and gender classification of faces and subsequent association with tracks. We introduce optional step for filtering faces based on face alignment. Additionally, we discuss how to produce a final outcome from multiple classification results.

We will be using CNNs for the classification task. We find the architectures *GoogLeNet* from [40] and *SSRNet* from [25] to be suitable for use in our algorithm. We chose these based on their speed and proclaimed accuracy, as we cannot properly evaluate them due to the dataset problems described in section 3.2.4.

4.4.1 Face Association

To propagate the age and gender information to the tracks, we need to associate them with the faces. We calculate a center point for each face's bounding box. We then (for each face) iterate over all tracks and assign the face to the all tracks eligible based on relative bounding box position.

Let $\xi_1, \xi_2, \xi_3, \xi_4 \in [0, 1]$ be hyperpameters, c_x, c_y the face center, x, y the track's bounding box top left corner coordinates and w, h its width and height. The track is eligible if

$$x + w \cdot \xi_1 \leq c_x \leq x + w \cdot \xi_2 \wedge y + h \cdot \xi_3 \leq c_y \leq y + h \cdot \xi_4.$$

The main problem with this approach is, we might assign multiple faces to a single track or a single face to multiple tracks. We do not think this issue can be solved without significant overhead. We believe, based on observations from our data, that reasonable choice of the hyperparameters presented above, along with assumed availability of multiple predictions for given tracks, make this issue tolerable.

Another possible approach would be getting better face position approximation from body pose estimation. While this would still leave some cases, where overlap happens (as is inevitable when projecting 3D world to 2D coordinates), it could improve the matching. We did not explore this idea further, as we have found body pose estimation to be too computationally challenging for real-time processing on our hardware.

4.4.2 Face Alignment

As discussed previously, face alignment is critical for explored face classification models. We found that many of the detected faces by selected models are not suitable for further classification. Furthermore, face alignment is a potentially interesting source of information for future expansion, as it allows us to approximate the target's field of view.

The best solution to the classification problems would be to train the model to handle differently aligned faces. However, this can be difficult in practice. Adding alignment-based validation for faces, described in this section, can be seen as a supplementary step.

One possible approach is to assume that most classifications are correct and solve this when combining multiple predictions. This introduces no additional overhead, as we should always assume that some predictions may be incorrect.

The models typically output a probability for the target classes, which can typically be interpreted as the model's confidence. Another approach might be to consider only detections with very high confidence, assuming that those should be well visible faces. This does not work in practice because models often output extremely high confidence for very low-quality detections, as we have found when experimenting in our dataset.

Facial landmarks localization is topic of [41], where authors present SOTA models for this task. Both 2D and 3D face alignment models are presented. We found the 2D version to be about two times faster while producing good results. The model's output is the location of 68 face landmark points. Visualisation of this output is shown in figure 4.2.

Based on these landmarks and some assumptions about human face geometry, we can quickly test if some selected landmarks are approximately at expected locations. We can measure the relative position of the eyes, nose, and mouth and combine this with the face's bounding box size. Based on this, we should be able to classify approximate face position and decide, for example, if the face image is frontal or from the side.

4.4.3 Final Output

Since we evaluate a video and not a single image, we expect to have multiple predictions for a single track. Some approach is needed to produce a final output of the algorithm. We designed the algorithm to finalize its decision once the track leaves the scene.

Some statistical function is needed to process the multiple predictions into a single outcome. We think reasonable choices are mode for the gender prediction and median for the age prediction.

If the given track has no associated prediction, we output its demographics as "unknown", as we have no prior knowledge about the class probabilities. Additionally, it might be reasonable to output "unknown" even when there

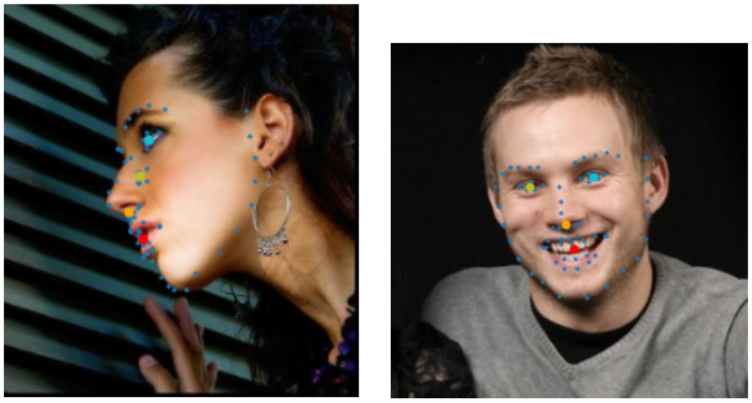

Figure 4.2: Visualisation of face landmarks, with certain landmarks highlighted. Images are from the LS3D-W dataset[41].

is data present. For example, if there is an equal or almost equal number of predictions for the different gender classes.

4.5 Algorithm Hyperparameters

The presented algorithm has a number of hyperparameters that need to be configured. This section presents their overview and used values in table 4.1. The values are presented mostly for illustrative purposes and as a potential starting point. Target scene, selected models and target goals must be considered when choosing the hyperparameters.

name	description	value
max_age	Tracks missed more times are considered lost.	7
max_age_active	Maximum track's age to be considered active.	1
min_hits	Minimum number of assigned detections to be considered confirmed.	5
feature_alpha	Weight for calculating new visual feature (higher means slower update).	0.8
motion_weight	Weight of the motion distance in contrast to visual feature distance (note that the motion distances are typically about an order larger).	0.02
iou_threshold	Minimal IOU to consider areas overlaping (when associating with IOU.	0.5
measurement_noise	Detected bounding box standard deviation.	100
process_noise_var	Kalman filter process variance.	10

Table 4.1: Selected algorithm hyperparameters.

4.6 Implementation

The pipeline is implemented in Python 3.6. We use *TensorRT* framework for optimized NN inference on the Jetson platform. *GStreamer* framework along with Jetson-specific NVIDIA plugins is used for efficient video stream processing. [30] was used as a basis for developing the pipeline, which proved especially helpful for efficient video processing and using the *TensorRT* framework.

Used Python libraries are *NumPy* for numerical computing and matrix operations, *Pandas* for data manipulation, *FilterPy* for Kalman filter implementation, *py-motmetrics* for MOT metric calculations and *OpenCV* for video and image processing.

4.6.1 Inputs and Outputs

The system should be usable with various input sources and produce output suitable for additional computer processing.

A standard protocol used in industrial and surveillance cameras is RTSP[42]. Handling input via this protocol allows the application to be easily connected to live streams from many types of cameras. The application also handles video input from the local filesystem to allow the processing of already recorded videos.

The application outputs location of all confirmed tracks for each frame along with their currently classified demographic information. The final demographic information is stored as a special event when a track leaves the scene. Output for the gender field is either male, female, or "unknown". For the age field, the output is either an age range or "unknown".

The application can optionally output the average frame rate and proportional time spend in various parts of the algorithms (time spent in detection, association, ...). Additionally, the pipeline can be provided with *ground truth* information for a given file and produce detailed statistics along with overall metrics.

Output can be either send periodically over the network or saved to the local filesystem. Data is saved in the JSON format, which is commonly used for data serialization.

Chapter 5

Experiments

This chapter presents experiments and the evaluation of our algorithm. The goal of the experiments is to compare various models and approaches in terms of quality and performance. Supplementary materials and code are available at `https://github.com/davidmasek/Algorithms-for-video-analysis-of-customer-behavior-in-front-of-retail-store`.

All evaluations are run on the Jetson Xavier NX platform[29], with the 15W and 4 core power configuration. Jetpack version is 4.1.1, CUDA version is 10.2 and TensorRT version is 7.1.3.

5.1 Tracker Evaluation

The main part of our pipeline is the MOT tracker. We evaluate its performance on our collected dataset, which consists of 2600 annotated frames with 49 unique objects and 7108 objects in total. We split the dataset into train and test subsets, with respective sizes of 1309 and 1291 frames.

Figure 5.1: A sample frame from the dataset with visualized ROI.

We run the pipeline without the demographics extraction. We evaluate only tracks and annotations inside the ROI, a rectangle covering the relevant

area in front of the shop. See figure 5.1 for visualisation. Results are presented in table 5.1. Final results are computed as an average of the evaluated sequences weighted by the number of frames. *Switch ratio* is the ratio between the number of identity switches and the number of objects in total (scaled by 1000 for readability). We consider the minimal frame rate for real-time processing to be 10 Frames Per Second (FPS).

YOLOv4[35] is trained on the *CrowdHuman*[36] dataset. *PeopleNet*[43] is trained on a proprietary NVIDIA dataset. The *SSD*[44] model is trained on the *COCO*[45] dataset and used with *InceptionV2*[46] as a feature extractor.

Model	MOTA↑	MOTP↑	Switch Ratio↓	FPS↑
PeopleNet	0.84	0.83	**8.6**	9.42
YOLOv4	**0.91**	**0.93**	8.7	**13.0**
SSD	0.53	0.73	25	12.8

Table 5.1: Tracker evaluation results with different detector models. The arrows indicate low or high optimal metric values.

Based on the results, we selected the *YOLOv4* model for use in the application. It has the best results in all categories except for the *switch ratio*, where the results are very close to the best. Based on the metrics and visual evaluation of the algorithm's outputs, we consider its performance satisfactory. We also consider the algorithm fast enough for real-time usage.

Apart from our dataset, we also experimented on the MOT17[14] dataset, which is popular in literature. We have found the dataset to be too different from our target environment. The dataset contains videos taken from different angles and (often) with a moving camera. Our work assumes a stationary camera position at an elevated viewpoint, and so we have decided not to use the MOT17 dataset for evaluation. However, it remains an interesting source of data for testing the algorithm's robustness.

5.2 Performance Optimization

This section evaluates multiple architectures from a performance standpoint. In particular, we are interested in performance gain from using the TensorRT optimization framework.

We evaluate the *SSRNet*[25] and *GoogLenet*[40] usable for age and gender classification. Further, we evaluate two *OSNet*[7] ReID architectures. As a baseline, we run the inference on CPU and GPU using *ONNX Runtime*[47], which is an open source machine learning framework. We then use the *TensorRT* framework to optimize the model and run inference. The results are presented in table 5.2.

We observe that using the *TensorRT* optimized models increases performance about three times. To test that the optimization does not signifi-

Architecture	CPU	GPU	TensorRT
SSRNet (batch size 1)	3.7 ms	4.02 ms	1.4 ms
SSRNet (batch size 32)	106 ms	8.85 ms	3.85 ms
GoogLeNet	144 ms	12 ms	3.17 ms
OSNet_x0.25	361 ms	41.8 ms	12.3 ms
OSNet_ain_x1.0	2.08 s	156 ms	53.8 ms

Table 5.2: Inference time comparison.

cantly decrease accuracy we evaluated the *SSRNet* model on the *MegaAge*[48] dataset. The test Mean Average Error (MAE) of the baseline version is 12.8, and the MAE of the optimized version is 14.4 (which is an error increase of 12.6%).

Based on our experiments, we conclude that the *TensorRT* framework can be used for significant performance gain while maintaining comparable results. However, it should be noted that the results may vary on different models or hardware.

Conclusion

The goal of this thesis was to design and implement a pipeline for tracking people in front of a retail store while also obtaining information about age and gender.

In the first two chapters, we presented necessary theoretical concepts and explored related works and existing approaches.

Next, we analyzed the requirements of the retail environment and potential solutions. To provide relevant evaluation, we collected a new dataset with cooperation from retail store owners. Collecting the dataset made us more aware of potential problems and helped in the algorithm design. Due to the current Covid19 epidemic, we were unable to include age and gender information in our dataset, which led us to focus on the tracking task and leave demographic feature extraction as a secondary objective.

Based on our analysis and research, we designed a MOT pipeline based on the DeepSORT algorithm presented in [15]. We modified the algorithm mainly in the association phase and extended it with the support for demographic information extraction. The tracking is based on associating new detections to existing tracks using the combination of two metrics. The first metric is IOU of Kalman filter predictions for tracks' locations and detections' bounding boxes. The second metric measures the visual similarity between detections and tracks.

Our pipeline is focused on performance and usability in the retail environment. We implemented the pipeline in Python using TensorRT framework for optimizations. The pipeline is optimized for the Jetson NX device, which has been chosen as a suitable device for production.

In the last chapter, we presented experiments comparing various models and evaluated our application on the collected dataset. We concluded that the algorithm provides satisfactory results for use in production and can run in real time.

The most relevant opportunities for future improvement are demographic information extraction and overall robustness. Specifically, it would be bene-

ficial to collect data from other locations and include the demographic information. We plan to continue working on the presented pipeline in cooperation with ImproLab laboratory at FIT CTU, with the goal of deploying the application in production.

Bibliography

[1] Shi, W.; Cao, J.; et al. Edge computing: Vision and challenges. *IEEE internet of things journal*, volume 3, no. 5, 2016: pp. 637–646.

[2] Russell, S.; Norvig, P. *Artificial Intelligence: A Modern Approach*. Pearson, fourth edition, 2020, ISBN 9780134610993.

[3] Karpathy, A. CS231n: Convolutional Neural Networks for Visual Recognition [online]. `https://cs231n.github.io/`, 2020, accessed: 2021–03-26.

[4] Hornik, K.; Stinchcombe, M.; et al. Multilayer feedforward networks are universal approximators. *Neural networks*, volume 2, no. 5, 1989: pp. 359–366.

[5] Bengio, Y.; LeCun, Y.; et al. Scaling learning algorithms towards AI. *Large-scale kernel machines*, volume 34, no. 5, 2007: pp. 1–41.

[6] Howard, A. G.; Zhu, M.; et al. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[7] Zhou, K.; Yang, Y.; et al. Omni-Scale Feature Learning for Person Re-Identification. 2019, `1905.00953`.

[8] Redmon, J.; Divvala, S.; et al. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 779–788.

[9] Werbos, P. J. Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, volume 78, no. 10, 1990: pp. 1550–1560.

[10] Goodfellow, I.; Bengio, Y.; et al. *Deep Learning*. MIT Press, 2016, `https://www.deeplearningbook.org`.

[11] Labbe, R. Kalman and Bayesian Filters in Python [online]. https://github.com/rlabbe/Kalman-and-Bayesian-Filters-in-Python, 2014, [Commit hash 24b9fb3cf756b3c765579decd624132efe7be374].

[12] Zhang, Y.; Wang, C.; et al. FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking. *arXiv preprint arXiv:2004.01888*, 2020.

[13] Bewley, A.; Ge, Z.; et al. Simple online and realtime tracking. *2016 IEEE International Conference on Image Processing (ICIP)*, Sep 2016, doi:10.1109/icip.2016.7533003. Available from: http://dx.doi.org/10.1109/ICIP.2016.7533003

[14] Milan, A.; Leal-Taixé, L.; et al. MOT16: A Benchmark for Multi-Object Tracking. *arXiv preprint arXiv:1603.00831*, 2016.

[15] Wojke, N.; Bewley, A.; et al. Simple Online and Realtime Tracking with a Deep Association Metric. In *2017 IEEE International Conference on Image Processing (ICIP)*, IEEE, 2017, pp. 3645–3649, doi: 10.1109/ICIP.2017.8296962.

[16] Ren, S.; He, K.; et al. Faster r-cnn: Towards real-time object detection with region proposal networks. *arXiv preprint arXiv:1506.01497*, 2015.

[17] Bochinski, E.; Eiselein, V.; et al. High-speed tracking-by-detection without using image information. In *2017 14th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS)*, IEEE, 2017, pp. 1–6.

[18] Milan, A.; Rezatofighi, S. H.; et al. Online Multi-Target Tracking Using Recurrent Neural Networks. 2016, 1604.03635.

[19] Sadeghian, A.; Alahi, A.; et al. Tracking The Untrackable: Learning To Track Multiple Cues with Long-Term Dependencies. 2017, 1701.01909.

[20] Kuhn, H. W. The Hungarian method for the assignment problem. *Naval research logistics quarterly*, volume 2, no. 1-2, 1955: pp. 83–97.

[21] Bernardin, K.; Stiefelhagen, R. Evaluating multiple object tracking performance: the clear mot metrics. *EURASIP Journal on Image and Video Processing*, volume 2008, 2008: pp. 1–10.

[22] Axis Communications. *AXIS FA1105 Sensor Unit*. 2010. Available from: https://www.axis.com/files/datasheet/ds_fa1105_t10089337_en_2010.pdf

[23] Sekachev, B.; Manovich, N.; et al. opencv/cvat: v1.1.0. Aug. 2020, doi: 10.5281/zenodo.4009388. Available from: https://doi.org/10.5281/zenodo.4009388

[24] Levi, G.; Hassner, T. Age and gender classification using convolutional neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, June 2015, pp. 34–42. Available from: https://osnathassner.github.io/talhassner/projects/cnn_agegender

[25] Yang, T.-Y.; Huang, Y.-H.; et al. SSR-Net: A Compact Soft Stagewise Regression Network for Age Estimation. In *IJCAI*, volume 5(6), 2018, p. 7.

[26] Karkkainen, K.; Joo, J. FairFace: Face Attribute Dataset for Balanced Race, Gender, and Age for Bias Measurement and Mitigation. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021, pp. 1548–1558.

[27] Pavlakos, G.; Choutas, V.; et al. Expressive body capture: 3d hands, face, and body from a single image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 10975–10985.

[28] Smilkov, D.; Thorat, N.; et al. TensorFlow.js: Machine Learning for the Web and Beyond. *arXiv preprint arXiv:1901.05350*, 2019.

[29] NVIDIA. Jetson Xavier NX Technical Specification [online]. [Accesed 21-04-14]. Available from: https://developer.nvidia.com/embedded/jetson-xavier-nx

[30] Yang, Y. FastMOT: High-Performance Multiple Object Tracking Based on YOLO, Deep SORT, and Optical Flow. Nov. 2020, doi: 10.5281/zenodo.4294717. Available from: https://doi.org/10.5281/zenodo.4294717

[31] Bergmann, P.; Meinhardt, T.; et al. Tracking Without Bells and Whistles. In *The IEEE International Conference on Computer Vision (ICCV)*, 2019, pp. 941–951.

[32] Blalock, D.; Ortiz, J. J. G.; et al. What is the State of Neural Network Pruning? 2020, 2003.03033.

[33] Molchanov, P.; Mallya, A.; et al. Importance Estimation for Neural Network Pruning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 11264–11272.

[34] Vanholder, H. Efficient inference with TensorRT [online]. 2016, [Accesed 21-04-14]. Available from: https://on-demand.gputechconf.com/gtc-eu/2017/presentation/23425-han-vanholder-efficient-inference-with-tensorrt.pdf

[35] Bochkovskiy, A.; Wang, C.-Y.; et al. Yolov4: Optimal speed and accuracy of object detection. *arXiv preprint arXiv:2004.10934*, 2020.

[36] Shao, S.; Zhao, Z.; et al. Crowdhuman: A benchmark for detecting human in a crowd. *arXiv preprint arXiv:1805.00123*, 2018.

[37] Wei, L.; Zhang, S.; et al. Person transfer gan to bridge domain gap for person re-identification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 79–88.

[38] Brown, R. G.; Hwang, P. Y. *Introduction to Random Signals and Applied Kalman Filtering*. Wiley and Sons, fourth edition, 2012, 143-147 pp.

[39] Bourgeois, F.; Lassalle, J.-C. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Communications of the ACM*, volume 14, no. 12, 1971: pp. 802–804.

[40] Szegedy, C.; Liu, W.; et al. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.

[41] Bulat, A.; Tzimiropoulos, G. How far are we from solving the 2D & 3D Face Alignment problem? (and a dataset of 230,000 3D facial landmarks). In *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 1021–1030.

[42] Schulzrinne, H.; Rao, A.; et al. Real time streaming protocol (RTSP). 1998.

[43] NVIDIA. PeopleNet [online]. [Accesed 21-04-14]. Available from: `https://ngc.nvidia.com/catalog/models/nvidia:tlt_peoplenet`

[44] Liu, W.; Anguelov, D.; et al. Ssd: Single shot multibox detector. In *European conference on computer vision*, Springer, 2016, pp. 21–37.

[45] Lin, T.-Y.; Maire, M.; et al. Microsoft coco: Common objects in context. In *European conference on computer vision*, Springer, 2014, pp. 740–755.

[46] Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, PMLR, 2015, pp. 448–456.

[47] Microsoft. ONNX Runtime [online]. [Accesed 21-04-29]. Available from: `https://github.com/microsoft/onnxruntime`

[48] Zhang, Y.; Liu, L.; et al. Quantifying Facial Age by Posterior of Age Comparisons. *arXiv e-prints*, 2017: pp. arXiv–1708.

Appendix A

Acronyms

AI Artificial inteligence.

CNN Convolutional Neural Network.

DeepSORT Simple Online and Realtime Tracking with a Deep Association Metric.

FPS Frames Per Second.

IOU Intersection over Union.

Jetson Xavier Jetson NX.

MAE Mean Average Error.

ML Machine learning.

MOT Multiple Object Tracking.

MOTA Multiple Object Tracking Accuracy.

MOTP Multiple Object Tracking Precision.

MSE Mean Squared Error.

NN Neural Network.

ReID re-identification.

RGB Red, Green, Blue.

ROI Region of Interest.

SORT Simple Online Realtime Tracking.

SOTA state-of-the-art.

Appendix B

Contents of enclosed CD

- `readme.txt` ..... the file with CD contents description
- `src` ..... the directory of LaTeX source codes of the thesis
- `thesis.pdf` ..... the thesis text in PDF format