TU WIEN Informatics

Pedestrian Crossing Prediction in the Context of Autonomous Driving

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Visual Computing

eingereicht von

Maximilian Pröll, B.Sc.
Matrikelnummer 01525029

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao. Univ. Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz
Mitwirkung: Dipl.-Ing. Dominik Schörkhuber

Wien, 16. Mai 2022

Maximilian Pröll	Margrit Gelautz

Informatics

Pedestrian Crossing Prediction in the Context of Autonomous Driving

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Visual Computing

by

Maximilian Pröll, B.Sc.
Registration Number 01525029

to the Faculty of Informatics

at the TU Wien

Advisor: Ao. Univ. Prof. Dipl.-Ing. Mag. Dr. Margrit Gelautz
Assistance: Dipl.-Ing. Dominik Schörkhuber

Vienna, 16th May, 2022

Maximilian Pröll	Margrit Gelautz

Erklärung zur Verfassung der Arbeit

Maximilian Pröll, B.Sc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 16. Mai 2022

Maximilian Pröll

Danksagung

Ich möchte mich bei meiner Betreuerin Margrit Gelautz bedanken, welche mir die Möglichkeit gegeben hat, an diesem Projekt mitzuwirken, und immer für Fragen zu dieser Arbeit zur Verfügung stand. Weiters möchte ich mich bei meinem Betreuer Dominik Schörkhuber bedanken, der nicht nur über das gesamte Projekt hinweg mit technischer Unterstützung für die Klärung von Implementierungsdetails zur Seite stand, sondern auch wertvolles Feedback für diese Arbeit gegeben hat. Auch möchte ich mich bei meiner Mutter bedanken, die mich über die Studienzeit finanziell unterstützt hat, was es mir ermöglicht hat, Vollzeit zu studieren.

Diese Arbeit wurde von dem Projekt SmartProtect (Projektnummer 879642) unterstützt, welches vom Bundesministerium für Klimaschutz (BMK) im Rahmen des FTI-Programms Mobilität der Zukunft finanziert und der Österreichischen Forschungsförderungsgesellschaft (FFG) abgewickelt wurde.

Acknowledgements

I want to thank my advisor Margrit Gelautz, who gave me the opportunity to work on this project, and was always available for questions related to this work. I also want to thank my advisor Dominik Schörkhuber, who supported me throughout the project with technical advise and the valuable feedback he gave me on this thesis. I also want to thank my mother, who supported me financially during my studies which allowed me to study full-time.

This thesis was supported by the project SmartProtect (Project Number 879642), which is funded through the Austrian Research Promotion Agency (FFG) on behalf of the Austrian Ministry of Climate Action (BMK) via its Mobility of the Future funding programme.

Kurzfassung

Autonomes Fahren ist ein wichtiges Gebiet der Computer Vision und des maschinellen Lernens, in welchem in den letzten Jahren maßgebliche Fortschritte gemacht wurden. Erst vor Kurzem hat sich der Fokus in der Wissenschaft weiter in Richtung der Vorhersage der Handlungen von Verkehrsteilnehmern bewegt, wobei im Besonderen die Vorhersage der Handlungen von vulnerablen Verkehrsteilnehmern von Interesse ist.

In dieser Arbeit evaluieren wir die Performance von state-of-the-art Computer Vision Systemen und Methoden des maschinellen Lernens für die Vorhersage von Straßenquerungen von Fußgängern (Pedestrian Crossing Prediction). Die meisten dieser Methoden nutzen Inputs, welche im Vorhinein generiert werden müssen. Wir untersuchen daher, wie sich das Ändern dieser Inputs und/oder der Netzwerkarchitektur auf die Vorhersagequalität auswirkt. Wir stellen fest, dass Faster R-CNN für die Detektion von Passanten besserere Ergebnisse liefert, wenn der Detektor nur mit Samples von Passanten trainiert wird, anstatt mit mehreren Klassen. Weiters zeigen wir, dass OpenPifPaf als Detektor für Passanten eine schlechtere Leistung als Faster R-CNN liefert. Bezüglich der Trajektorienvorhersage von Passanten (Pedestrian Trajectory Prediction) führen wir eine qualitative und quantitative Evaluierung der Vorhersageergebnisse des DTP Netzwerks durch. Das Netzwerk verwendet für die Vorhersage ausschließlich den optischen Fluss des Input Bildes. Die Ergebnisse zeigen, dass dieses Trajektorienvorhersagenetzwerk keine befriedigenden Ergebnisse liefert. Unsere Experimente mit Pedestrian Crossing Prediction zeigen, dass die Performance von SFRNN verbessert werden kann, wenn verlässlichere Posenschätzungen (Pose Estimations) von Passanten als Input verwendet werden. Für diese Experimente generieren wir die Posen mit HRNet. Durch die Verwendung dieser Posen erhöhen wir die durchschnittliche Genauigkeit (accuracy) über die JAAD und PIE Datensätze, im Vergleich zu den ursprünglichen Ergebnissen, um 10 Prozentpunkte. Außerdem ändern wir SFRNN so ab, dass der optische Fluss des Bildes anstatt der Posen verwendet wird. Diese Netzwerkversion erreicht eine um 8 Prozentpunkte höhere Genauigkeit über beide Datensätze. Dies deutet darauf hin, dass der optische Fluss auch ein nützlicher, alternativer Input für Pedestrian Crossing Prediction sein kann. Weiters stellen wir fest, dass durch die Verwendung von zusätzlichen Inputs, für das von uns evaluierte Netzwerk, keine Leistungssteigerung im Hinblick auf die Vorhersagequalität erreicht wird, sondern die Leistung sogar verschlechtert wird. Im Allgemeinen zeigt sich, dass die Vorhersagequalität zwischen den Datensätzen schwankt, was darauf hindeuten könnte, dass größere Datensätze verwendet werden sollten.

Abstract

Autonomous driving is an important application area of computer vision and machine learning where significant advancements have been achieved in the last years. Only recently, the research in this field has shifted towards the behavior prediction of traffic actors and road users, especially regarding vulnerable traffic participants.

In this thesis, we evaluate the performance of state-of-the-art computer vision and machine learning methods for pedestrian crossing prediction. Most of these methods rely on input data based on precomputed features. Hence, we investigate in particular how altering existing networks' inputs and/or network structure can improve the performance. We find that the performance of Faster R-CNN is increased if its training relies only on pedestrian samples rather than multiple classes. We further show that OpenPifPaf does not perform as well as a dedicated object detector, such as Faster R-CNN, for detecting pedestrians. Regarding pedestrian trajectory prediction, we qualitatively and quantitatively evaluate DTP, which only uses optical flow as an input. We find that this simple trajectory predictor does not yield satisfactory results for this particular task. Our experiments on pedestrian crossing prediction show that the performance of SFRNN can be improved by using more reliable human pose estimations as input. In particular, we generate pose estimations with HRNet. Using these inputs, we achieve a 10 percentage points higher accuracy score over the JAAD and PIE datasets in comparison to the original, evaluated network. We also alter SFRNN to use optical flow instead of poses. This network increased the average accuracy over all datasets by 8 percentage points in comparison to the original version of SFRNN, which indicates that optical flow can be a useful alternative input for the task of pedestrian crossing prediction. We observe that adding more inputs to the crossing prediction network does not further increase the performance, but rather decreases it. Generally, we discover that the prediction performance fluctuates with different datasets, indicating that the used training data set should be augmented in size.

Contents

CHAPTER 1

Introduction

The ambition to create autonomous cars has been around for many decades. However, technical means to achieve this goal were very limited in the past. With increasingly more powerful computers and the rise of deep learning in the last decade, many new possibilities in the field of computer vision arose. Hence, the biggest advancements in autonomous driving have been made in this time period.

Nevertheless, many computer vision methods for autonomous driving still lack the accuracy and reliability to be introduced into the mass market. Therefore, the adoption of fully autonomous cars will most certainly still take quite some time. However, these systems may be used for driver assistance, since lower accuracy and reliability is less critical if the driver is still controlling the car. These systems are therefore already in use to prevent accidents and increase road safety.

Early work in this field focused on simple tasks like lane and distance-keeping. With the rise of deep learning, the number of possible computer vision applications increased significantly. Moreover, deep learning based methods began to outperform classical computer vision methods and therefore now present the current state of the art.

Deep learning became popular in the last decade, as it made remarkable advances in different areas of machine learning possible [Dea20], one of which is computer vision. In the field of computer vision, specifically object detectors made striking leaps in accuracy and performance in the last 20 years [ZSGY19]. Therefore, the focus of research gradually shifted to more challenging problems like behavior prediction of humans. Especially in the context of autonomous driving, it is necessary to gauge the possible actions of other road users, in order to prevent accidents. The ability to "read" people is crucial for a car to be fully autonomous, as humans usually communicate with subliminal signs (i.e., eye contact, gait, body posture etc.) to indicate, if they want to interact with the car (e.g., show intent of crossing the street).

Only recently, the task of behavior prediction in automotive applications is also considered in computer vision research. As a result, this topic is not yet well covered in the literature which presents a major motivation for this thesis, in which we want to explore the development of selected behavior prediction algorithms, and analyze how these methods can be applied to pedestrian crossing prediction in the context of autonomous driving.

1.1 Problem Statement

Humans naturally have the ability to read people to some extent by their body language. Especially in road traffic, drivers need to understand the intention of pedestrians without the possibility to communicate verbally. Hence, in the context of road traffic, conventions for non-verbal communication exist (e.g., hand gestures, holding eye contact). Moreover, traffic actors also communicate subconsciously. For example, the gait and/or body posture may indicate, if a person is just about to cross the road, or if the person is slowing down and will most probably stop at the side of the road. Additionally, the pedestrian may not pay attention at all to the road traffic. In that case, it is also important for the pedestrian's safety that the driver realizes the unawareness of the pedestrian. Therefore, reading peoples' behaviors and intentions is crucial for driving a car safely, especially in urban areas, where many encounters occur.

Figure 1.1 shows an example sequence from the Joint Attention in Autonomous Driving (JAAD) dataset (datasets are discussed in Section 2.1) with behavioral annotations for the pedestrian and the ego vehicle. Actions like making hand gestures or looking towards the driver may be used to determine the intention of the pedestrian. Machines lack this ability to read people and recognize their actions. The research focus on this problem has increased with recent developments in the field of deep learning.

State-of-the-art methods for pedestrian crossing prediction often rely on several different input features and associated computer vision methods at the same time. This results in complex systems, which are computationally expensive to run and output unreliable predictions. This motivates our approach to scrutinize different methods for crossing prediction and evaluate how the overall performance of such networks can be improved by appropriate altering of the inputs.

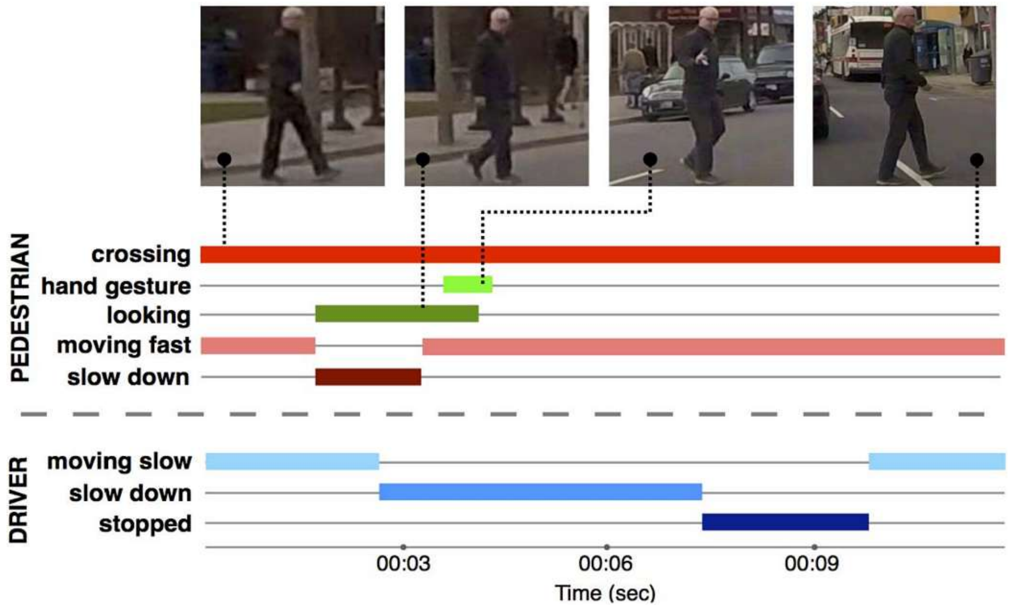


Figure 1.1: Behavioral annotations present in the JAAD dataset. Timestamps represent the sequence of the events that took place during the crossing action. [RKT17]

1.2 Aim of the Work

The main goal of this thesis is to explore and evaluate current state-of-the-art methods for pedestrian crossing prediction, which can be used to avoid collisions with vulnerable road users and therefore increase road safety. This includes experiments with the two groups of action prediction and trajectory prediction algorithms. Furthermore, potential improvements (e.g., by altering and optimizing input features) of these methods are explored and evaluated.

As most algorithms which are targeted to the task of pedestrian behavior prediction rely on cropped images of pedestrians, we further aim to evaluate the feasibility of using a human pose estimation network to generate pedestrian bounding boxes and compare the performance of the evaluated pose estimation network to a general object detector. Additionally, we investigate possible performance improvements for the task of bounding box generation.

1.3 Structure of the Thesis

In Chapter 2, we start with an overview of current state-of-the-art methods for action and trajectory prediction, including the commonly used datasets specialized for this task. State-of-the-art object detection is also discussed, as it forms the foundation for generating bounding boxes used as inputs for many pedestrian behavior prediction algorithms. Additionally, state-of-the-art methods for optical flow estimation and pose estimation are explored, as they are also used frequently as inputs for pedestrian behavior prediction algorithms. In Chapter 3, we discuss the methodology of the work, describing the selected approaches, used datasets, the working environment, used tools and the evaluation metrics. In Chapter 4, we explain the neural networks used for our experiments and describe the adaptions made to the networks. The experiments and evaluation results are presented in Chapter 5. Finally, Chapter 6 comprises a discussion of the results and gives an outlook on possible future developments.

CHAPTER 2

Background and State of the Art

This chapter gives an overview of current state-of-the-art methods for pedestrian behavior prediction. First, we explore commonly used datasets to train neural networks on this specific task. Next, we explain how pedestrian behavior prediction methods work and which types of inputs are used. Among the most commonly used features for pedestrian behavior prediction are cropped images of pedestrians, pedestrians' poses, optical flow and semantic segmentations of the input scene. These inputs are usually generated in a pre-processing step, using dedicated neural networks. We therefore shortly summarize how these inputs can be generated. In particular, we discuss methods for object detection, human pose estimation and optical flow.

2.1 Datasets for Pedestrian Behavior Prediction

As the interest in pedestrian behavior prediction increased over recent years, the need for datasets specialized on this task increased to the same extent. An early dataset capturing the behavior of pedestrians was published in 2014 by Kooij et al. [KSFG14], called the Daimler Pedestrian Path Prediction Benchmark Dataset. The dataset consists of 58 sequences recorded using a stereo camera mounted behind the windshield of a vehicle. Each sequence only contains one pedestrian with the intention to cross the street, but features different levels of criticality where "critical" and "non-critical" situations are distinguished from each other. On average, the sequences are 7.15 seconds long and have all been annotated the Time To Event (TTE), expressing the point in time when the pedestrian is going to cross the street. Furthermore, the dataset contains annotations for the pedestrians' head orientations, which may be used to classify the pedestrians' situational awareness towards the ego-vehicle.

More recent datasets were published by Rasouli et al., namely the Joint Attention in Autonomous Driving (JAAD) [RKT17] and Pedestrian Intention Estimation (PIE) [RKKT19] datasets. With the JAAD dataset, the focus lies on pedestrian and driver behavior at the point of crossing. The dataset contains 346 video clips, each of which is 5-10 seconds long. These videos were recorded in North America and Eastern Europe and represent typical urban driving scenes in different weather conditions like rain and snow. All pedestrians are annotated with bounding boxes and occlusion tags, which makes this dataset suitable for pedestrian detection. 25% of those pedestrians are also annotated with behavioral tags, which capture the pedestrians' behavior around the crossing event. These behavioral tags can be categorized in *precondition* (standing, moving slow, moving fast), *attention* (looking, glancing) and *response* (stop, clear path, slow down, speed up, hand gesture, nod). This subset of the JAAD dataset will be denoted by JAAD_{beh} from here on. The entire JAAD dataset containing all pedestrian samples will be denoted as JAAD_{all}. An example frame from the JAAD dataset, containing all annotations, is depicted in Figure 2.1.

For the purpose of intention estimation and trajectory prediction, the JAAD dataset has a number of drawbacks, such as the absence of accurate ego-vehicle information like the speed, the short length of the videos which are discontinuous chunks, and the fact that the majority of pedestrian samples with behavioral annotations have the intention of crossing. To this end, the PIE [RKKT19] dataset was introduced. It contains over 6 hours of footage recorded in typical traffic scenes with an on-board camera. To provide ego-vehicle information, the speed, heading direction and GPS coordinates were also recorded. There are around 300k labeled video frames with 1842 pedestrian samples. The intent annotations for the pedestrians were obtained by crowdsourcing. Each participating person had to watch a short sequence from the PIE dataset and decide, if the observed pedestrian has the intention to cross the street. The average of all the answers were used

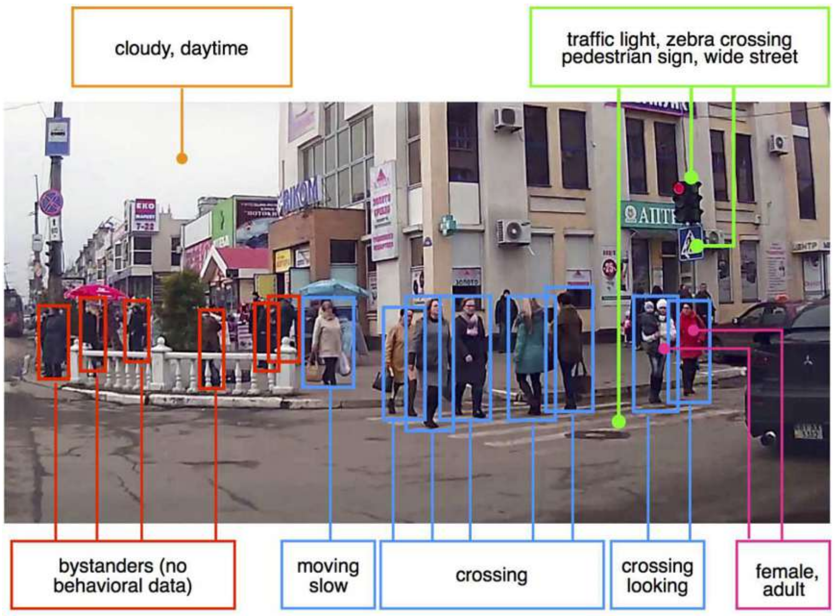

cloudy, daytime
traffic light, zebra crossing pedestrian sign, wide street
bystanders (no behavioral data)
moving slow
crossing
crossing looking
female, adult

Figure 2.1: Example of annotations provided in the JAAD dataset. [RKT17]

to annotate the pedestrian samples with an *intention probability*.

In 2020, the Trajectory Inference using Targeted Action priors Network (TITAN) dataset [MDC20] was published by the Honda Research Institute. As the name suggests, the dataset is targeted on training networks for predicting trajectories of road users. It contains over 10 hours of video material at 60Hz, all of which was recorded in Tokyo. There are roughly 400k annotated pedestrians, 150k annotated 4-wheeled vehicles and 100k annotated 2-wheeled vehicles. Especially the high number of 2-wheeled vehicles in the dataset create an opportunity to generate more robust algorithms for detecting bicycles and motorcycles, since they are usually underrepresented in other driving-datasets. Furthermore, there are 50 labels, where $\approx 2/3$ of the labels are used for pedestrian annotations. Each pedestrian is annotated with an atomic action (e.g., walking, standing, running, sitting). Moreover, a *simple-context* (e.g., cleaning, walking on the side, exiting a building, biking) or *complex-context* annotation (e.g., loading, unloading, getting in vehicle, getting out of vehicle) may be provided when applicable.

2.2 Pedestrian Behavior Prediction

Within the field of pedestrian behavior prediction, the most common tasks are action/intent prediction (Section 2.2.1) and trajectory prediction (Section 2.2.2). In the context of autonomous driving, the estimation of the *risk potential* of a vulnerable road user is of utmost importance. Most work focuses on the crossing prediction of the pedestrian [RRL+18] to estimate the risk potential of that person. Practically speaking, if the person is predicted to cross the street in front of the ego vehicle, the risk potential is high. We therefore also mainly focus on the task of crossing prediction. However, with a predicted trajectory, the crossing prediction of a person is implicitly given and the additional information of a person's path is useful for autonomous driving. Hence, both approaches will be discussed in the following subsections.

2.2.1 Pedestrian Action and Intent Prediction

Action and intent prediction with methods of computer vision were largely made possible due to rapid advances in deep learning. State-of-the-art methods nowadays usually rely on deep learning to accomplish this task. Early work in this field mostly focused on action recognition, i.e., the sole classification of the action performed by a single or multiple persons in a video sequence. E.g., in 2014, Karpathy et al. [KTS+14] presented different approaches for action recognition using deep learning methods. They specifically explore different approaches to fuse spatio-termporal data inside a neural network in such a way that the issue of extensively long periods of training time is mitigated. The authors explored four methods of fusing information within the network: *Single Frame, Late Fusion, Early Fusion* and *Slow Fusion* (see Figure 2.2). *Single Frame* is understood as a reference baseline solution, as only one frame is used as an input. With *Late Fusion*, two frames are processed independently with two *Single Frame* networks, where parameters, which are 15 frames apart, are shared between the networks. As the two streams merge

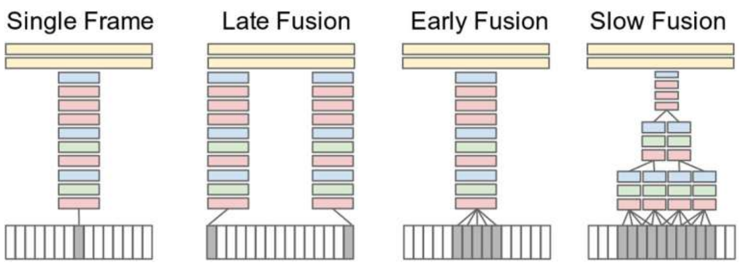


Figure 2.2: Different approaches for fusing information over temporal dimension through a Convolutional Neural Network (CNN). Red, green and blue boxes indicate convolution, normalization and pooling layers respectively. In the Slow Fusion model, the layers depicted side-by-side (columns) share parameters. [KTS+14]

into a fully connected layer, neither single frame stream can detect motion on its own, but global motion characteristics are computed inside the fully connected layer by comparing outputs of both layers. *Early Fusion* is similar to *Single Frame* in that regard, as it only uses one stream. However, the information across an entire time window is immediately combined on the pixel level. *Slow Fusion* combines the approaches by slowly fusing temporal information through the network in such a way, that higher layers have access to progressively more global information in both spatial and temporal dimensions. In [KTS+14], the authors found *Slow Fusion* to work best, but also stated that *Single Frame* performs well in comparison.

Simonyan and Zisserman further explored the possibilities of two-stream convolutional networks [SZ14]. In contrast to Karpathy et al. [KTS+14], they do not use a context image for the second stream, but the optical flow of the image sequence. The proposed method is based on the *two-stream hypothesis*, which states that the human visual cortex contains two pathways: namely the ventral stream (object recognition) and the dorsal stream (recognition of motion) [GM92]. The final result is obtained by averaging the outputs of the two streams.

In the more specific field of pedestrian crossing prediction, most methods use recurrent architectures [RKT20, RKKT19, LPWS20, KRT21, YZY+22] to predict the crossing action of a pedestrian. More recently, multi-task-learning networks have been proposed, which not only predict the crossing event, but also additional outputs like the future trajectory [RRL21, RGB+20]. Other multi-task-learning networks exist, where the network detects the pedestrians on the image and predicts different attributes [SHN17, MCPA20, ZAZC20] or other features like the pedestrians' poses [RMA21]. In [PRC+19], the authors built a multi-task model that predicts the pedestrian's bounding box and action in an end-to-end manner.

Most of these works use poses as at least one of their inputs [FL18, RGB+20, KRT21, RKT20, LAC+20, GPB+20, GMB+18, YZY+22]. Other common inputs are the pedestrians' bounding boxes [RKT20, KRT21, YZY+22, RKKT19], scene semantics [RGB+20, RRL21, YZY+22] and the cropped images of the pedestrians [RKT20, RKKT19, LPWS20].

For example, in 2020, Rasouli et al. [RKT20] introduced a network for predicting pedestrians' crossing intentions, denoted by Stacked with multilevel Fusion Gated Recurrent Units (SF-GRU). The prediction relies on five types of inputs, namely the pedestrian's appearance, the pedestrian's local context referring to the pedestrian's surrounding, the pedestrian's pose, the pedestrian's 2D bounding box location and the speed of the ego-vehicle. The authors compared the performance of SF-GRU to five different network models, including a multi-stream Gated Recurrent Unit (GRU) (the features are processed separately using different GRUs and only combined at the end in a dense layer for prediction) and a hierarchical GRU (similar to the multi-stream GRU, but all the hidden states of every GRU are passed to another GRU where its last layer is used for prediction). The network outperforms the other compared network models, achieving an accuracy of 84% on the PIE dataset.

Kotseruba et al. [KRT21] compared different network architectures for predicting pedestrians' actions in their PedestrianActionBenchmark and proposed a Recurrent Neural Network (RNN) using the pedestrians' bounding boxes, poses, local context and the ego-vehicle speed as inputs, which also surpasses the performance of SF-GRU.

2.2.2 Trajectory Prediction

The research on trajectory prediction was mainly focusing on surveillance in the past [AGR$^+$16, CHMC21, DOL20, SKS$^+$19]. However, due to the rising interest in autonomous driving over the recent years, a considerable amount of papers describing trajectory prediction of road users were published. We mainly focus on the work done by Styles et al. [SRS19, SGS20], as their work proposes methods for the specific task of pedestrian trajectory prediction in the context of autonomous driving and therefore is most relevant for our use case.

In [SRS19], the authors present a pedestrian trajectory forecasting system using optical flow, namely Dynamic Trajectory Predictor (DTP). They specifically propose to use optical flow features as input, since manually labeling poses is cumbersome and time-consuming. Moreover, automatic generation of poses with pose estimation networks (Section 2.5) is challenging and error prone.

The goal of DTP is to predict the future centroid of a pedestrian's bounding box given the last m optical flow frames of said bounding box before the prediction. Instead of directly predicting the future location, the authors propose to predict a *compensation term* C_t, which corrects the errors in the trajectory predicted using the constant velocity assumption. They argue that the model is first initialized to a strong baseline (if $C_t = 0$ the constant velocity assumption holds) and then fine-tunes predictions on training examples where the constant velocity assumption does not hold and results in errors.

The network is trained on the JAAD dataset using the last m optical flow frames before the prediction. Optical flow is directly predicted on pedestrians' bounding boxes using FlowNet 2.0 (see Section 2.4). The frames are resized to 224×224. As horizontal and vertical flow are output separately by the used network, the frames are stacked, resulting in an input vector of size $2m \times 224 \times 224$. This input vector is then fed to a modified ResNet, which learns to predict the compensation term C_{t+n}.

Since dense (per-frame) bounding box annotations of pedestrians are expensive to obtain by hand, in [SRS19], Styles et al. further propose to generate trajectories automatically: The authors use YOLOv3 [RF18] and Faster R-CNN [RHGS16] (see Section 2.3) for detecting pedestrians and Deep SORT [WBP17] to generate tracks over multiple frames.

Building on top of DTP, in 2020, Styles et al. [SGS20] propose their Spatio-Temporal Encoder-Decoder (STED) for Multiple Object Forecasting. STED does not only take visual features into account (past optical flow), but also the temporal features extracted by a GRU from past object bounding boxes (see Figure 2.3 for an overview of the architecture). Furthermore, they introduce the *Citywalks* dataset[1], which consists of over 200k video frames and 3.5k unique pedestrian trajectories. The footage was recorded in 21 cities from 10 European countries in a variety of weather conditions.

The CNN for generating the features from optical flow is adapted from DTP. The number of optical flow frames is chosen to match a time frame of one second. The CNN then generates an optical flow feature vector ϕ_f, which is then passed to the fully connected layer (note that there is a mistake in Figure 2.3, which is taken from the original publication: ϕ_f needs to be before FC-2, not after[2]). As optical flow estimation is applied on the pedestrians' images, cropped by their bounding boxes, the optical flow captures both object motion and ego-vehicle motion and are encoded in ϕ_f.

The bounding boxes are encoded as 8-dimensional vectors where each bounding box object at every timestep t is defined as follows: $B_t = (x_t, y_t, w_t, h_t, v_t^x, v_t^y, \Delta w_t, \Delta h_t)$, where x, y, w, h denote the coordinates of the bounding box's centroid, the width and the height. v^x, v^y denote the velocity in x and y direction. The changes in width and height

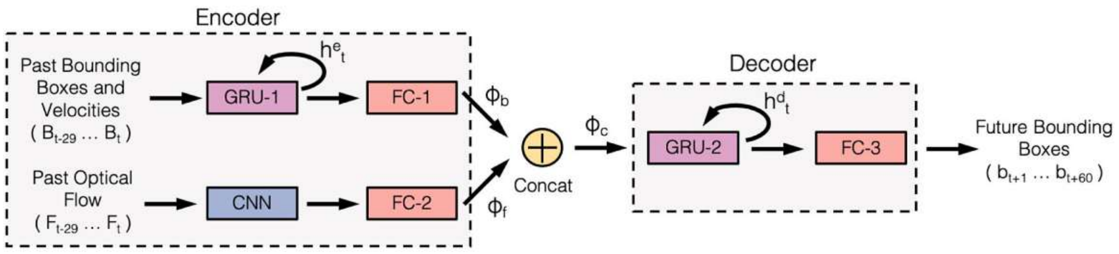

Encoder
Past Bounding Boxes and Velocities ($B_{t-29} \dots B_t$)
GRU-1
h^e_t
FC-1
Past Optical Flow ($F_{t-29} \dots F_t$)
CNN
FC-2
ϕ_b
ϕ_f
Concat
ϕ_c
Decoder
GRU-2
h^d_t
FC-3
Future Bounding Boxes ($b_{t+1} \dots b_{t+60}$)

Figure 2.3: Overview of the STED architecture: The network can be divided into two main components, namely the encoder and decoder. For feature encoding, the past bounding boxes (passed through a GRU) and the past optical flow frames (passed through the CNN known from DTP) are concatenated. The encoder subsequently takes the encoded feature vector ϕ_c as input and outputs predicted object bounding boxes for the next two seconds using another GRU. [SGS20]

[1]`https://github.com/olly-styles/Multiple-Object-Forecasting`
[2]`https://github.com/olly-styles/Multiple-Object-Forecasting/issues/10`

over the observed time frame are denoted by Δw and Δh. For each observed timestep t, a GRU (GRU-1 in Figure 2.3) takes B_t as input and outputs an updated hidden state vector h_t^e. This update is repeated for every timestep of the input sequence and results in a hidden state vector h_t^e at the final timestep, which encompasses and summarizes the entire sequence of bounding boxes. This vector is then transformed to a 256-dimensional feature vector ϕ_b (see FC-1 in Figure 2.3).

In STED, instead of ResNet18 as for the original DTP implementation, a ResNet50 is used. The optical flow feature encoder is trained for 10k iterations. Eventually, the weights are frozen to use the flow encoder as a fixed feature extractor. In their evaluation, compared to DTP, STED achieves a 1.3% lower average distance error on the Citywalks dataset.

2.3 Object Detection

Object detection is one of the fundamental problems in computer vision, hence it has received a lot of attention in the last 20 years. Therefore, there are many object detection implementations which achieve state-of-the-art performance within their field of application [ZSGY19].

Object detectors using CNNs can be divided into 2 main categories, namely two-stage-detectors and single-stage detectors. The former group uses a separate stage to generate region proposals, which are then used in the second stage to classify the object and apply bounding box regression. One of the most prominent two-stage-detectors is Faster R-CNN, which was introduced in 2015 and still performs comparably well [RHGS16]. On the other hand, single-stage detectors detect objects in one pass, without an intermediate step. Prominent examples of one-stage-detectors are YOLO (You Only Look Once) [RDGF16], SSD (Single Shot multibox Detector) [LAE+16] and RetinaNet [LGG+17]. While they are faster compared to two-stage-detectors, they are usually not able to perform as well when it comes to accuracy or struggle with the detection of very small objects.

Since Faster R-CNN is still widely used in this field, we briefly discuss its working principle in the following. As mentioned earlier, the main problem with two-stage detectors is the high computational cost, which is mainly caused by the Region Proposal Network (RPN). To mitigate this problem, Ren et al. [RHGS16] proposed to share the full-image convolutional features from the detection network with the RPN, which enables nearly cost-free region proposals. Their work builds up on Fast R-CNN [Gir15], which is used as the detection network. The authors construct an RPN on top of the convolutional feature maps from Fast R-CNN by adding additional convolutional layers that simultaneously regress region bounds and objectness scores at each location on a regular grid. The resulting network was named Faster R-CNN and can be trained end-to-end.

To train the RPN and Fast R-CNN at the same time, the authors developed a specific training procedure: The fine-tuning is alternated between the region proposal and the object detection task. This scheme showed to quickly converge and produce a unified network with convolutional features shared between both tasks.

Since the introduction of Faster R-CNN, it is widely used for different object detection tasks in various environments. In 2015, Faster R-CNN has been the foundation for several 1st-place entries in the ILSVRC[3] and COCO competitions. Up to this day, Faster R-CNN still performs comparably well [KSP20] and therefore remains a popular method for object detection.

2.4 Optical Flow

Optical flow aims to calculate the motion field of a frame from a video sequence. The goal is to estimate the motion between consecutive frames, caused by the movement of objects or the camera itself. Hence, optical flow is a method, which is prominently used in computer vision for different tasks (e.g., motion detection for surveillance, 2D-to-3D conversion, video segmentation, obstacle detection and avoidance). For observation of humans in the automotive context, the optical flow also encodes the motion of pedestrians, which can be utilized to reason about the behavior of pedestrians. The most well-known differential methods for calculating optical flow are Lucas-Kanade [LK81] and Horn-Schunck [HS81]. The Lucas-Kanade method is a local method, which means that for calculating the optical flow, it assumes neighboring pixels to have a constant brightness over multiple frames. Horn-Schunck, on the other hand, assumes an overall smoothness of the resulting optical flow vector field and hence is a global method. While these global methods produce smoother vector fields, they are more sensitive to noise [BWS05].

As deep neural networks have become the state-of-the-art for many computer vision tasks in recent years, research in the field of optical flow has also shifted towards deep learning. One of the most prominent work in this field was done by Dosovitskiy et al. with FlowNet [DFI+15]. In their work, they propose an end-to-end trainable CNN which is able to estimate optical flow, given two input images. Furthermore, the synthetic Fyling Chairs dataset was generated, containing 3D models of chairs placed on top of random background images from the real world. This enables them to train a CNN with a reasonable amount of data.

The main problem with CNNs is that the resolution gets quite low towards the end of the network due to the convolutions. The last layer in the network is the most sensitive to category-level semantic information (e.g., the class value an item belongs to). On the other hand, lower layers (i.e., layers, which are closer to the input of the network pipeline) contain more detailed information about the image like edges, precise locations and illumination. For tasks like object detection, where pixel-perfect positions of the bounding boxes may not be necessary, the low resolution towards the end of the network is not a problem. However, the task of estimating optical flow requires pixel-accurate prediction. Other tasks which rely on per-pixel predictions are semantic segmentation,

[3]https://www.image-net.org/challenges/LSVRC/

depth prediction, edge detection and keypoint prediction (see Section 2.5). Different methods exist to solve this issue. The simplest solution is to apply a CNN on smaller parts of the image in a sliding window fashion [LH20, GB14]. While this method works well in many situations, it has drawbacks like lacking knowledge of global properties, which may result in misclassifications. Moreover, this method is computationally expensive, even with optimized implementations, where intermediate feature maps are reused [GB14]. Another simple approach is to upsample all the feature maps to a desired resolution and afterwards concatenate them [HAGM15]. From this stack of feature maps, a feature column may be extracted, which can be used to predict the value of interest.

Dosovitskiy et al. [DFI+15] solved this problem by developing their own *refinement* pipeline for the FlowNet network. Within the refinement part, they upconvolve all the coarse feature maps, in order to transfer more high-level information to the fine prediction. Moreover, the upconvolution results are then concatenated with the contractive part of the network to refine the resulting optical flow output image. The upconvolution essentially is the opposite of pooling, which may also be called *unpooling*. Different approaches to this method exist [ZTF11, ZF14, LSD15, DTSB15]. Specifically for FlowNet, this upconvolution is done four times, where the resolution is doubled at each convolution. Moreover, two network architectures were tested, namely FlowNetSimple (FlowNetS) and FlowNetCorr (FlowNetC). FlowNetS uses a standard CNN architecture with a single stream. The two input images are stacked together and fed to the network. With FlowNetC however, two separate, yet identical streams exist where each stream processes one image. The two streams are then combined at a later stage, where a correlation layer is used to match the feature maps generated by the streams.

FlowNet demonstrated that optical flow estimation can be learned with a simple CNN. However, traditional state-of-the-art approaches outperformed FlowNet regarding the quality of the optical flow estimation. Particularly small displacements in the input images created problems such as noisy artifacts in the estimated flow fields. To remedy this issue, Ilg et al. [IMS+17] introduced FlowNet 2.0, which improved the estimation error by more than 50% in comparison to the original FlowNet. The authors experimented with stacking multiple FlowNetC/S architectures, in order to refine the results and improve the accuracy of FlowNet. The results are refined as follows: First, the second input image is warped by the optical flow estimation of the preceding network via bilinear interpolation to $\tilde{I}_2$. This result is passed to the subsequent network. The error of the optical flow estimation is calculated as $e = \|I_2 - I_1\|$, which is also used as input for the subsequent network. With that information, the FlowNet 2.0 network refines the optical flow estimation throughout the pipeline (see Figure 2.4 for visualization of the stacked architecture of FlowNet 2.0). Furthermore, the authors specifically refine a FlowNetS to work well for small displacements by making it deeper through exchanging the 7x7 and 5x5 kernels with multiple 3x3 kernels. Additionally, they add convolutions between the upconvolutions to reduce noise present in FlowNetS with small displacements.

With FlowNet 2.0, Ilg. et al. demonstrate several improvements to the original FlowNet which result in accuracy that is on par with state-of-the-art methods while simultaneously

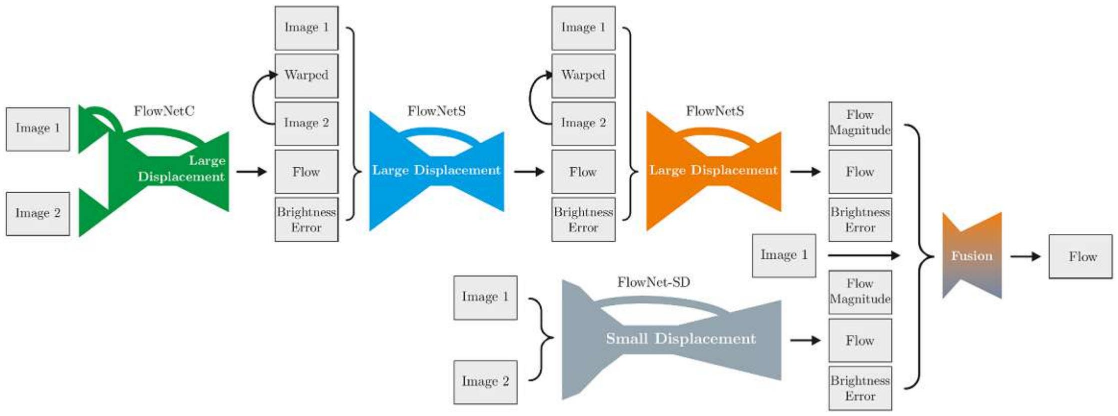


Figure 2.4: Schematic view of complete FlowNet 2.0 architecture: Multiple stacked FlowNets are used to achieve good optical flow estimation results for large and small displacements in input images. Inputs from the preceding network are concatenated (indicated by braces) and then fed to the subsequent network. The *brightness error* is the difference between the first and the second image, where the second image is warped with the previously estimated flow. [IMS$^+$17]

running with shorter inference time.

2.5 Human Pose Estimation

Human pose estimation describes the task of finding different keypoints of human bodies in images and videos. With these keypoints, human poses can be inferred, which to some extent encode the intent or the current action performed by this person. This information is useful for action and intent prediction. We therefore shortly discuss the state-of-the-art of human pose estimation.

OpenPose was introduced in 2017 by Cao et al. [CSWS17]. It became a highly popular implementation for human pose estimation as it achieves accurate results while being fast enough for real time applications. Moreover, OpenPose was the first open-source realtime system for multi-person 2D pose detection and hence, it is now widely used in the field of computer vision. In 2019, a new version of the original 2017 paper appeared, containing several additional contributions, like a slightly modified network architecture, which achieves better results on the COCO dataset [CHS+21]. OpenPose is a *bottom-up* approach, meaning that the keypoints are directly predicted on the whole input image and connected at a later stage using association scores via Part Affinity Fields (PAF). PAFs are a set of 2D vector fields which encode the location and orientation of limbs over the image domain [CSWS17]. With the *top-down* approach on the other hand, persons are detected first and the keypoint detection only happens within the bounding box of that person. The downside of a top-down approach is that a single-person pose estimator is run on every bounding box, resulting in increased computational cost in proportion to the number of people on the image. The overall pipeline of OpenPose works as follows: With the input image, the system creates a set of confidence maps, one for each human pose keypoint. Furthermore, a set of 2D vector fields of part affinities is created, encoding the degree of association between the body parts. Finally, the confidence maps and the affinity maps are parsed by greedy inference to output the 2D keypoints for all people in the image.

As it can be seen in Figure 2.5, the network of OpenPose is split into two branches.

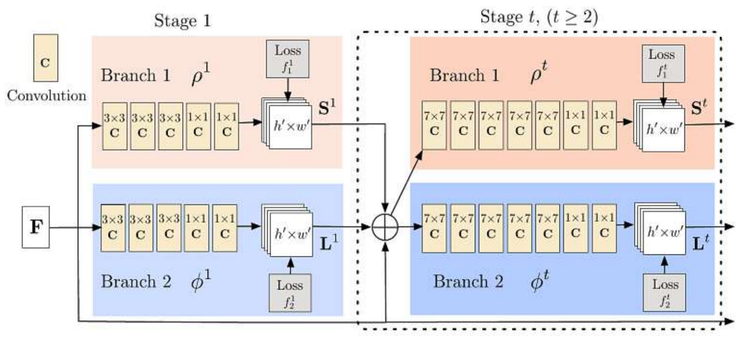


Figure 2.5: OpenPose architecture with two-branch multi-stage CNN. Each stage in the first branch predicts the confidence maps S^t and each stage in the second branch predicts the part affinity fields L^t. [CSWS17]

The image is first transformed to a set of features F by passing it through a VGG-19 network [SZ15]. The feature set is passed to both network branches. The beige branch is responsible for creating the detection confidence maps, whereas the blue branch is responsible for creating the part affinity fields. To refine the predictions, the results from both branches are concatenated, along with the original image features F. In the subsequent stages, this procedure is repeated using a slightly varied branch architecture using bigger convolution kernels. With the PAFs, the network is able to estimate, which key points from the confidence maps belong to which limb. At the same time, these body limbs are associated to one person, by comparing all possible key point matches to PAF predictions. The optimal association is hence reduced to a maximum weight bipartite graph matching problem, as only single pairs of body parts are considered.

Kreiss et al. [KBA19] took the idea from [CSWS17] one step further and extended the notion of fields in pose estimation to go beyond scalar and vector fields to *composite* fields. Similar to OpenPose, PifPaf uses a shared network with two heads. The first head predicts the fields containing the body part joints. However, it does not only contain the position, but also the size of this joint. These fields are called Part Intensity Fields (PIF). More formally, every point in a PIF is composed of a scalar component for confidence, a vector component which points to the closest body part of the particular type and another scalar component for the size of the joint. The fusion of these components is depicted in Figure 2.6. The other network head predicts the associations between the body joints, called Part Association Field (PAF) with a composite structure. These fields contain the associations between each pair of connected joints. Hence, 19 PAFs exist. At every output location, PAFs predict a confidence value, two vectors, whereas each one is pointing to one of the two parts, and two widths for the spatial precision of the regression. A visualization of a PAF for left hips and shoulders is shown in Figure 2.7. In comparison to OpenPose, PifPaf achieves a 4.9% higher Average Precision (AP) on the COCO 2017 test-dev dataset [KBA19]. In 2021, Kreiss et al. introduced OpenPifPaf [KBA21], which includes several new contributions which improve the performance even further. For example, they propose to remove hidden keypoints (i.e., occluded keypoints) if they are

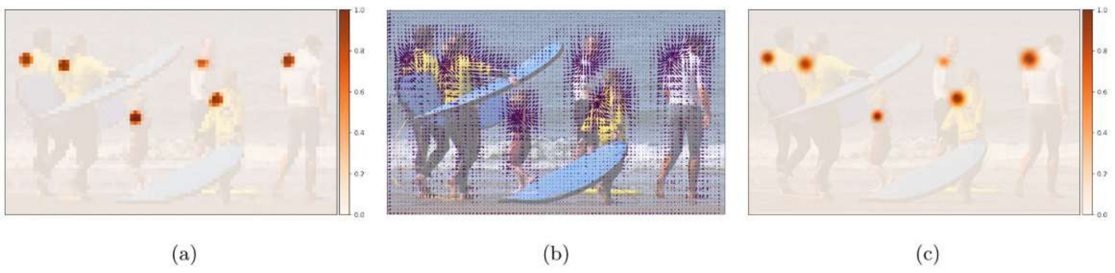


Figure 2.6: Visualization of components of the PIF for the left shoulder (in total, 17 PIFs exist, as 17 keypoints are used with PifPaf). Subfigure (a) shows the confidence maps and (b) visualizes the vector field pointing to left shoulders. Subfigure (c) shows the fused PIF, where the confidence and joint scale are combined. [KBA19]

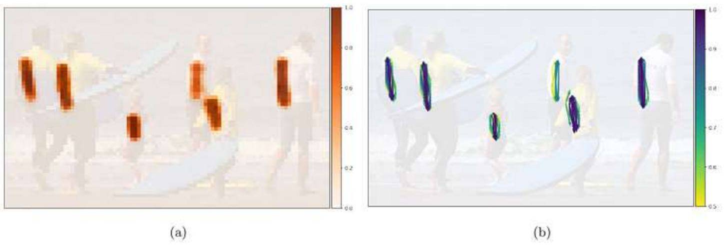


Figure 2.7: Visualization of components of the PAF which associates left shoulders with left hips. The figure depicts one of 19 PAFs. At every location of the PAF, two vectors pointing to the hip and shoulder are generated. The confidence of these associations a_c are depicted in (a). The vector components for $a_c > 0.5$ are depicted in (b). [KBA19]

close to a visible keypoints. In their experiments, this step has shown to improve pose reconstruction. Furthermore, they extend the PAFs with joint-scale components and call it Composite Association Field (CAF). With these additions, OpenPifPaf achieves a 9.1% higher AP on the COCO 2017 test-dev dataset in comparison to OpenPose, while having a lower inference time [KBA21]. Moreover, OpenPifPaf introduces a Temporal Composite Association Field (TCAF), which enables the network to track humans over a sequence of frames and achieve state-of-the-art results on the PoseTrack 2017 and 2018 dataset [KBA21]. This TCAF may also be used to improve the overall detection accuracy on video sequences, as false detections can be corrected with this additional temporal information.

Examples of bottom-up pose estimators are the baseline implementation provided by [XWW18], which we refer to as Simple Pose, and High Resolution Net (HRNet) [SXLW19]. The goal of Simple Pose is to provide a simple, yet effective baseline implementation for human pose estimation. The authors use a ResNet, where a few deconvolutional layers are added over the last convolution stage. By default, three deconvolutional layers with batch normalization [IS15] and ReLU activation [KSH12] are used. This approach achieves good results, where the biggest network reaches an AP of 73.7% on the COCO test-dev dataset. With HRNet, the goal is to increase the accuracy of the predicted keypoint heatmap by maintaining a high resolution of the feature maps throughout the network. Other existing networks for pose estimation are connecting high-to-low resolution subnetworks in series, where the resolution is raised at the end of the network to output the actual keypoint prediction on the input image (as it is also done in the Simple Pose network). To this end, the authors of HRNet propose to connect high-to-low resolution networks in parallel rather than in series. As a result, the resolutions of the parallel subnetworks of a later stage also contain the resolution of the previous stage. This enables the network to maintain the high resolution instead of recovering it through a low-to-high resolution subnetwork. In comparison to OpenPose, HRNet achieves a 13.7% higher AP on the COCO test-dev set with an AP of 75.5%.

CHAPTER 3

Methodology

The primary goal of pedestrian behavior prediction in the context of autonomous driving is to estimate the risk potential of a collision with a pedestrian. For this purpose, we focus on crossing prediction, where the goal is to predict if a pedestrian will cross the road at a certain point in time in the future. In this chapter, we discuss the methodology of the conducted experiments. This includes the considerations on using certain approaches, the used datasets, development environment and evaluation metrics.

3.1 Selected Approaches

We have identified two main approaches in the literature for the task of crossing prediction: **pedestrian action prediction** and **trajectory prediction**. With action prediction, the type of action is predicted explicitly (e.g., "crossing" or "not crossing"). With trajectory prediction on the other hand, the future path of a person is predicted, which implicitly contains the aforementioned crossing prediction. In particular, if the pedestrian's predicted trajectory crosses the road, the predicted action for the pedestrian is "crossing". In our experiments, we mainly focus on directly predicting the crossing action with a pedestrian action prediction network. However, to evaluate the feasibility of using a trajectory predictor to predict the crossing action, or even using the predicted trajectory as input for other action prediction networks, we also experiment with trajectory prediction. Additionally, since pedestrian action and trajectory prediction algorithms heavily rely on different inputs, which are derived from the pedestrians' bounding boxes, we further evaluate the **detection of pedestrians** on datasets for autonomous driving with a general object detector. In our evaluation environment, the pedestrians' bounding boxes are provided by the datasets, so that no additional complexity in the evaluation of our pedestrian action prediction network is introduced. Therefore, the experiments on pedestrian detection are conducted separately. An overview of the experiments and their goals is given in the following:

- **Pedestrian action prediction:** Pedestrian action prediction networks usually rely on multiple inputs (see Section 2.2.1). Our goal is to analyze different input types and evaluate which inputs show favorable performance for the chosen task of pedestrian action prediction. We further conduct experiments on how these inputs can be improved and how the performance in terms of accuracy of the evaluated action prediction network is affected. The experiments are conducted within the Pedestrian Action Benchmark (PAB) framework (Section 3.2) to compare performance improvements to other action prediction methods present in the benchmark. We base our experiments on the Stacked with multilevel Fusion Recurrent Neural Network (SFRNN) architecture, which provides state-of-the-art performance according to the benchmark, while using a comparably simple architecture, which allows us to experiment with different input features and feature alterations. The way we alter the network is discussed in Section 4.2.2.

- **Trajectory prediction:** We explore the performance of DTP (Section 2.2.2) on the JAAD dataset. We chose DTP, as it also uses a simple architecture and, therefore, the number of input features needed for the evaluation is also limited. The goal is to analyze the generated trajectories produced by DTP and to evaluate, if these trajectories may be used for crossing prediction, or may also be used as an input for other action prediction networks.

- **Pedestrian detection:** For our experiments on pedestrian detection, we decided to use Faster R-CNN, as it still performs comparably well [KSP20] and implementations are available in many deep learning frameworks. We evaluate how Faster R-CNN specifically performs in the context of autonomous driving and analyze if the performance can be improved if training is restricted to a single class. Since pedestrians' poses are among the most popular features for pedestrian action prediction (see Section 2.2.1), we will further explore if OpenPifPaf, being a pose estimation network, may replace a general object detector. This is particularly interesting, since OpenPifPaf also comes with a tracking feature, which enables tracking pedestrians over multiple frames. If OpenPifPaf shows satisfying performance on pedestrian detection, it could therefore not only replace an object detector, but also a tracking algorithm, which is needed to generate the input sequences for action prediction networks.

3.2 Datasets

Our work makes use of the Pedestrian Action Benchmark (PAB)[1], which is a standardized benchmark for evaluating the performance of networks targeting the prediction of crossing actions. It was introduced in 2021 by Kotseruba et al. [KRT21] with the goal to make different methods for pedestrian action prediction comparable, as different methods use different evaluation methods. With action prediction, a certain timeframe before the

[1] `https://github.com/ykotseruba/PedestrianActionBenchmark`

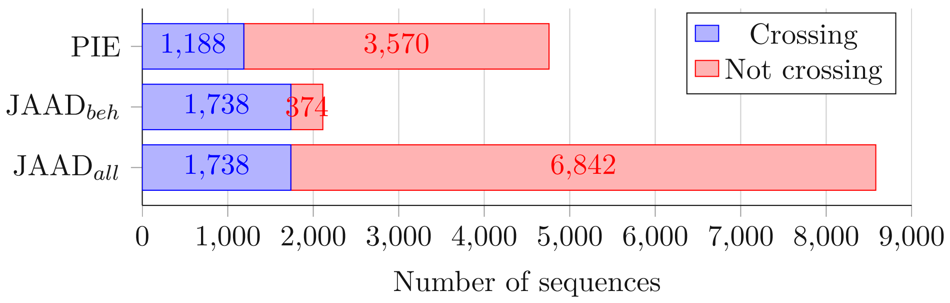


Figure 3.1: Proportions of crossing and non-crossing samples of datasets used for training in the Pedestrian Action Benchmark.

actual prediction needs to be observed. However, this timeframe varies between works. For example, some authors may use only 0.5 seconds of the video sequence before the action prediction, others may use an observation timeframe of 2 seconds. This not only makes comparing the evaluation results difficult, but also skews the performance: Action prediction becomes progressively easier the closer the actual crossing event [KRT21]. When the pedestrian is already crossing, most models achieve high prediction accuracy. Therefore, the PAB was proposed, which not only contains standardized evaluation procedures, but also a number of baseline and state-of-the-art models for pedestrian crossing prediction. Hence, all our experiments on pedestrian action prediction are conducted within the PAB framework.

The objective within the PAB is to predict the pedestrian's action (crossing/not crossing) at a future timestamp t, given an observed sequence of length m. Regarding our experiments inside the PAB framework, we use the standard settings where the observation length is 16 frames and the Time To Event (TTE) is set to 30 frames. The latter means that, if a video sequence has a framerate of 30 frames per second, the predicted action would occur 1 second in the future *after* the observation timeframe.

For training and evaluating the networks, we use the JAAD and PIE datasets (Section 2.1), since the PAB also depends on these datasets. With the JAAD dataset, two subsets exist: The JAAD$_{beh}$ and the JAAD$_{all}$ dataset. The JAAD$_{beh}$ dataset only contains the pedestrians with a behavioral annotation, whereas the JAAD$_{all}$ contains all pedestrians which have been annotated with a bounding box. The proportions of crossing and non-crossing samples (each sample is a sequence containing the data of 15 frames) used for training are depicted in Figure 3.1: 20%, 82% and 25% of samples are crossing samples for the JAAD$_{all}$, JAAD$_{beh}$ and PIE dataset, respectively.

3.3 Development Environment and Choice of Tools

For the purpose of training and evaluating our networks, a dedicated machine was used. The machine is equipped with a Nvidia RTX 3090 with 24GB of VRAM. This amount of VRAM enables us to train multiple networks at once. Furthermore, the machine is equipped with 32GB of RAM and an AMD Ryzen 9 3900X CPU. The operating system is Ubuntu 20.04, as most available network implementations were written for Linux.

Regarding the software stack, all the implementations are written in Python. As our pipeline includes code from different authors, different machine learning libraries are used in our project, including Keras[2], TensorFlow[3] and PyTorch[4]. This, however, can create problems, as the reference implementations of research papers often require different versions of the same dependencies. To mitigate this problem, we use Conda[5], a Python package management system which downloads and installs Python dependencies and makes sure, that versions are compatible with each other. Moreover, virtual environments can be created with Conda, which enables us to quickly test a certain implementation outside of our pipeline.

3.4 Evaluation Metrics

This section describes the evaluation metrics used for the experiments conducted in Chapter 5.

3.4.1 Metrics Used in the Pedestrian Action Benchmark

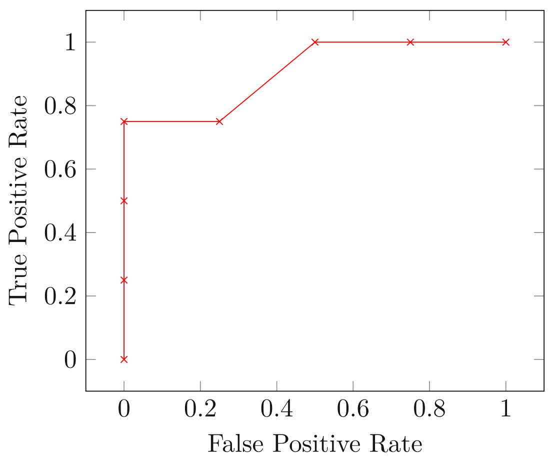
For evaluating the performance of the action prediction networks, the metrics Accuracy (ACC) , Area Under Curve (AUC), F1, Precision and Recall are used. The *accuracy* is denoted as

$$ACC = \frac{TP + TN}{TP + TN + FP + FN}$$

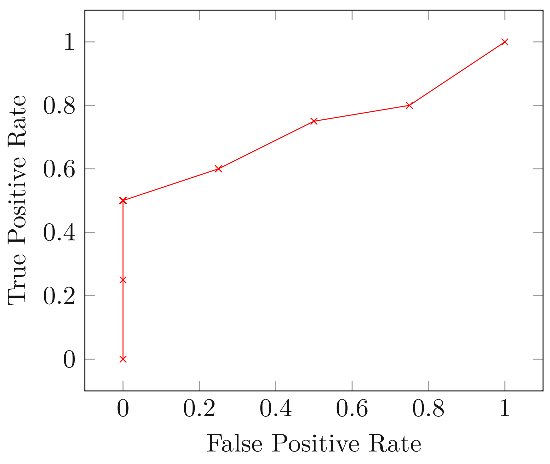
where TP and TN denote the number of True Positive and True Negative samples, respectively. Analogously, FP and FN stand for False Positive and False Negative, respectively.

The AUC defines the Area Under the ROC (Receiver Operating Characteristic) Curve. The ROC curve indicates how well a classifier can separate positive and negative samples (i.e., the quality of the threshold defined by the classifier to separate the samples). The curve connects all the points produced by the confusion matrices for each evaluated threshold. Every threshold generates a different true positive and false positive rate. These points may be connected in a graph, yielding the ROC curve. Two example ROC curves are depicted in Figure 3.2. When the AUC is calculated, different classifiers may be compared with each other. The classifier producing the highest AUC can be considered the best classifier, as it has a better measure of separability. In this depicted example, the classifier yielding the ROC curve on the left is the better classifier.

[2]https://keras.io/
[3]https://www.tensorflow.org/
[4]https://pytorch.org/
[5]https://docs.conda.io/en/latest/



(a) Depending on the real world use case, different false positive rates would be acceptable. For the lowest false positive rate, the threshold which yielded the point $(0, 0.75)$ would be considered best.



(b) When comparing this ROC curve with the ROC curve from the left, it becomes evident that this classifier performs worse, as the AUC is lower.

Figure 3.2: Example ROC Curves: The ROC curve summarizes the false positive and true positive rates of the confusion matrices produced by every threshold. In this fictional case, 8 thresholds for two different classifiers were evaluated, yielding 8 points in each chart.

Next, we will discuss *precision* (also sometimes referred to as positive predictive value) and *recall* (also known as sensitivity). The precision indicates how many positive identifications were correct. E.g., if a person detector detects 7 out of 10 persons, but predicts 15 additional bounding boxes where no person is present, the precision would be $7/(7 + 15)$. The corresponding formula is

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

Recall, on the other hand, measures the proportion of actual positives of all samples classified as positive. Regarding the example from above, the FN value is 3, since $10 - 7 = 3$, yielding a recall value of $7/(7 + 3)$. The related formula is defined as

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

Depending on the use case (e.g., a person detector for autonomous driving, where not detecting a person could be fatal), a higher recall at the cost of a lower precision may be preferred.

Since precision and recall have an inverse relationship, a metric which combines these two to better evaluate the overall performance of classifiers is useful. This metric is called

$F1$ score and calculated according to

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall}$$

The highest possible $F1$ score is 1.0 (indicating a perfect precision and recall) and the lowest possible score is 0.

3.4.2 COCO Evaluation Metrics

The COCO evaluation metrics[6] are especially useful for analyzing the performance of object detectors. They take different IoU (Intersection over Union) combinations and area sizes of the ground truth bounding boxes into account. For the IoU, the range 0.5:0.95 indicates that the evaluation values (either from AP or Average Recall (AR)) are averaged across 10 IoU values (0.5, 0.55, 0.6, ..., 0.95). If only one IoU value is given (e.g., $AP^{IoU=0.5}$), this IoU value indicates the minimum IoU which is used to decide if a detected object is considered for the AP or AR calculation. Furthermore, different area sizes of the ground truth bounding boxes are defined. *Small* bounding boxes have an area of $< 32^2$ pixels, *medium* bounding boxes have an area of $\geq 32^2$ and $< 96^2$ pixels and *large* bounding boxes have an area $\geq 96^2$ pixels. When *all* is indicated for the area size, all ground truth bounding boxes are used for the evaluation.

[6]https://cocodataset.org/#detection-eval

CHAPTER 4

Detection and Prediction Networks

This Chapter summarizes the experiments conducted on pedestrian detection, trajectory prediction and crossing prediction. We describe the methodology on generating the human pose estimation and optical flow inputs. Furthermore, the adaptions made to our networks, which are necessary to conduct certain experiments, are explained in detail.

4.1 Pedestrian Detection

The goal of our experiments on pedestrian detection is to evaluate the feasibility of using a bottom-up pose estimation network instead of a general object detector to generate bounding boxes of pedestrians. We use Faster R-CNN as a baseline general object detector, since it performs comparably well [KSP20] and implementations are available in many machine learning libraries. We use the implementation of Faster R-CNN which is provided in the PyTorch library[1].

Since the PIE dataset does not contain annotations for every visible pedestrian, it is not suitable for training an object detector like Faster R-CNNN. The reason is that while training an object detector, the unlabeled objects are treated as background, which impairs the performance of the network [PKSS21]. We instead train the network using the BDD100K dataset[2], which in total contains over 130k person instances [YCW$^+$20]. The PyTorch framework provides interfaces for creating custom datasets. After having implemented the necessary dataset class, we can also control which labels should be used

[1]https://pytorch.org/vision/stable/_modules/torchvision/models/detection/faster_rcnn.html
[2]https://www.bdd100k.com/

for training. This is necessary to evaluate, if the network performance can be improved, if only a single label (the pedestrian label) is used for training.

We chose the pose estimation network OpenPifPaf for this experiment and compare its detection performance to Faster R-CNN. OpenPifPaf also comes with a tracking module, which enables the pose estimator to track the detected poses over multiple frames. In theory, the pose estimator could therefore replace both a dedicated object detector and a dedicated tracking algorithm. To evaluate the detection performance of OpenPifPaf, we first extract the predicted bounding boxes and convert them to the COCO format. Then, the outputs are evaluated with the COCO API based on the COCO evaluation metrics.

4.2 Pedestrian Action Prediction

The goal of our experiments on pedestrian action prediction is to increase the performance of the evaluated action prediction network. In particular, we experiment with different input types and also seek to improve the inputs originally used by the network. These subsections describe the general procedure on how the experiments on pedestrian action prediction are conducted. We first chose a baseline neural network implementation for predicting pedestrian crossing actions. One of the baseline implementations provided in the PAB is SFRNN, which we have already discussed in Section 2.2.1. It should be noted that SF-GRU was renamed to SFRNN in the PAB.

SFRNN was introduced in 2020 by Rasouli et al. [RKT20]. The network predicts if a pedestrian will cross the street given the observed context up to some time m. The prediction relies on five types of inputs: the local context referring to the visual appearance of the pedestrian, the pedestrian's surrounding, the pedestrian's pose, the 2D bounding box location and the speed of the ego-vehicle. The network comprises five GRUs, each of which takes a sequence of features of one of the aforementioned input types and the hidden states of the GRU in the previous level as input (see Figure 4.1 for a visualization of the architecture). The features are gradually fused into the network according to the complexity of the features, whereby each sequence of features consists of m observations. The features are fused into the network as follows: pedestrian appearance $c_p^{1:m}$ (features generated by pre-trained VGG16 network), surrounding context $c_s^{1:m}$ (features generated by pre-trained VGG16 network), pedestrian's pose $p^{1:m}$ (generated by OpenPose 2.5 network), bounding boxes $b^{1:m}$ and ego-vehicle speed $s^{1:m}$. The local context encompasses the pedestrian's appearance and surrounding. The former is captured by cropping the image to the size of the 2D bounding box. The latter is obtained by extracting a region around the pedestrian by scaling up the 2D bounding box. To only extract the features from the surrounding, the pedestrian appearance is suppressed by setting the pixel values in the original bounding box coordinate to neutral gray. The actual features are then generated by passing the images through a pre-trained VGG16 network [SZ15], resulting in the feature vectors $vc_p^{1:m}$ and $vc_s^{1:m}$. The authors of SFRNN generated the poses by passing the cropped images of the pedestrians to OpenPose. OpenPose generates 18 body joints coordinates in 2D image space, which are then normalized and

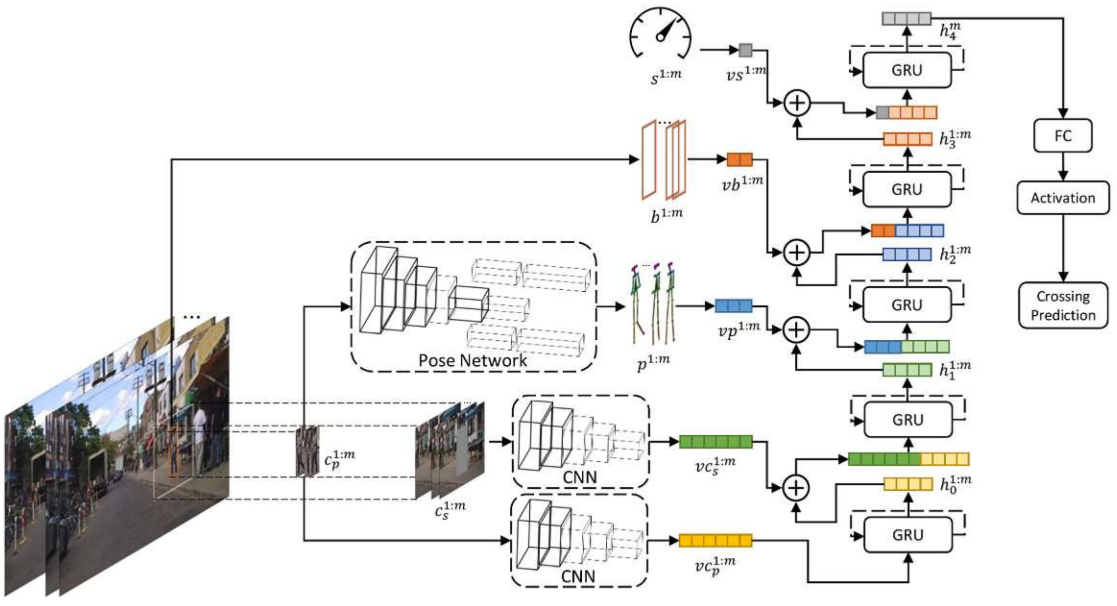


Figure 4.1: Overall architecture of SFRNN: The network is comprised of 5 GRUs, one for each input type. The features are gradually fused into the network according to the complexity of the features, whereby each sequence of features consists of m observations. [RKT20]

concatenated into a 36D feature vector $vp^{1:m}$. The 2D bounding boxes are transformed into a relative displacement from the initial position at the beginning of the observed sequence, forming the feature vector $vb^{1:m}$. This may be seen as the pedestrian's velocity. The ego-vehicle speeds are provided by the dataset and are converted to the feature vector $vs^{1:m}$, representing the car's speed at each time step of the sequence.

Since SFRNN uses a simple architecture and achieves state-of-the-art results in the PAB, we apply the same architecture for our experiments. As the network uses multiple inputs, it enables us to experiment with different input combinations and alterations. The overall experiment procedure is depicted in Figure 4.2. We first analyze the inputs' quality and decide which inputs could be improved. Then, the network is trained and evaluated on the JAAD and PIE datasets with the standard settings provided by the PAB. According to the evaluation results, the input generation and/or the network itself may be altered. This procedure is repeated several times until a conclusion can be drawn. In Section 4.2.1 we describe how we take the decision on changing certain inputs. Also, the feature generation itself is discussed, including the generation of qualitatively better poses and optical flow images of the pedestrians. In Section 4.2.2, we explain how the features are processed by SFRNN.

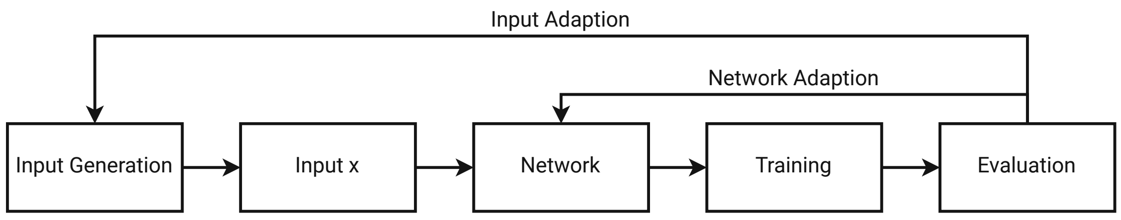


Figure 4.2: Flowchart of procedure used for experiments: Multiple inputs are generated for the network. Based on the evaluation results, either the input generation or the network itself is adapted.

4.2.1 Input Feature Generation

Standard SFRNN uses five inputs: the pedestrian's local context, local surrounding, pose, bounding box, and the vehicle's speed. Originally, the poses were generated by OpenPose, which are provided by the authors of the PAB. Upon inspection of the used pre-generated poses, we hypothesize that they are the only unreliable input, since there are many poses where the prediction is either incorrect or even completely missing (in that case, a zero vector is used for the pose input). Our assumption is supported by other works which found the provided poses to degrade the network performance [LAI$^+$21, LAC$^+$20]. Therefore, we generate new poses to evaluate their impact on the action prediction regarding the input quality. For that purpose, OpenPifPaf, Simple Pose and HRNet are used. Furthermore, since optical flow is widely used in the field of behavior prediction [SRS19, SGS20, SZ14, TDT12, LCCZ21, LZD20], we experiment with it as an alternative input feature for the SFRNN network. The generation of these features is discussed in the following subsections.

Human Pose Estimation

The PAB generates sequences for training, testing and validating the networks. Within this generation process, the poses can also be generated and cached for future network trainings. In the case of OpenPifPaf, we directly use the prediction API[3]. First, the image is cropped based on the bounding box and then the pose estimator is applied on this cropped image. Since OpenPifPaf is a bottom-up pose estimator, it can theoretically output more than one pose for an image. For the sake of simplicity, we always use the first output from the prediction. With Simple Pose, the procedure is the same. However, as Simple Pose is a top-down pose estimator, there is only one pose output per image. For our experiments, we use the GluonCV Simple Pose implementation[4], whereby the best performing pre-trained model is used. As with the original poses from OpenPose, we normalize the newly computed poses from OpenPifPaf and Simple Pose between 0 and 1 based on the bounding box dimensions.

[3]`https://openpifpaf.github.io/predict_api.html`
[4]`https://cv.gluon.ai/model_zoo/pose.html`

Examining the method for cropping pedestrian images within the PAB, we find that the resulting images are not optimal for pose estimation, as the nearest neighbor method is used as sampling method. This results in highly pixelated images, for which pose estimation is a difficult task (see Section 5.3 for the evaluation results). To prevent pixelation of the cropped images, we use the Lanczos filter [Duc79] instead, which is provided in the Python PIL Image library[5]. The Lanczos filter is defined as:

$$L(x) = \begin{cases} sinc(x)sinc(\frac{x}{a}) & \text{if} -a < x < a, a \neq 0 \\ 1 & \text{if} x = 0 \\ 0 & \text{else} \end{cases}$$

where $sinc(x)$ is defined as $\frac{sin(\pi x)}{\pi x}$.

We further conduct experiments with HRNet, since models trained on the BDD100K dataset are publicly available[6]. We hypothesize that HRNet trained on this dataset will perform best for our task, as the other pose estimation networks are trained on the COCO dataset, which does not provide data samples from road scenes. We use the HRNet implementation provided in the MMPose toolbox[7]. Since the used HRNet implementation expects input data in the COCO format, the annotations for the PIE and JAAD datasets need to be converted. The annotation files for the JAAD and PIE dataset are given in XML files, which contain all pedestrian tracks for a video. Each track contains all bounding boxes of a pedestrian derived from tracking over multiple frames. A sample bounding box annotation is depicted in Listing 4.1.

Listing 4.1: Sample bounding box annotation from the PIE dataset

```
<box frame="1014" keyframe="0" occluded="1" outside="0" xbr="933.78"
    xtl="921.50" ybr="849.78" ytl="800.25">
<attribute name="id">1_1_2</attribute>
...
</box>
```

Each bounding box annotation contains, among other parameters, the frame number of the video, the coordinates and multiple attributes like the pedestrian id. However, annotations in the COCO format are structured differently: The JSON annotation file contains, first, an array which contains the information for every image (file name, dimensions, id), and second, an array containing the actual annotations, where each annotation contains data like the keypoints, bounding box, annotation id and image id. For the conversion, a Python script was written, which outputs a JSON file for every set in the PIE and JAAD datasets. An example output is depicted in Listing 4.2.

Listing 4.2: Sample excerpt from a JSON file containing data in the COCO format

[5]https://pillow.readthedocs.io/en/stable/reference/Image.html#PIL.Image.resize

[6]https://github.com/SysCV/bdd100k-models/tree/main/pose

[7]https://github.com/open-mmlab/mmpose

```
images:
[0]:
file_name: "set01-video_0002-05150.jpg"
heigt: 1080
width: 1920
id: 1
...
annotations:
[0]:
segmentation: [],
keypoints: [...],
num_keypoints: 0,
area: 6834.470599999994,
iscrowd: 0,
image_id: 1,
bbox:
[235.12, 726.27, 51.91, 131.66],
category_id: 1,
id: "1_2_23"
...
```

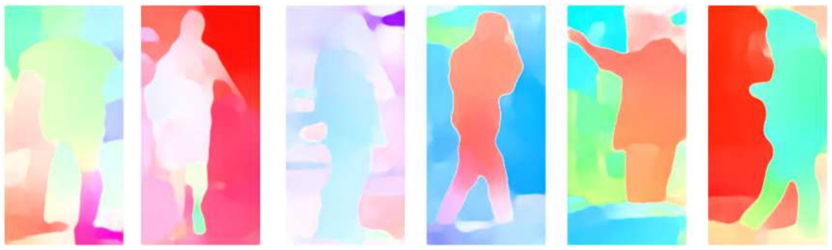
Similar to Simple Pose, HRNet is also a top-down approach. Consequently, it uses the annotated bounding boxes from the JSON files and crops the video frames internally to generate the pose estimations. Eventually, HRNet outputs a JSON file, which contains all pose estimations for every video frame. Inside the PAB, the JSON files are loaded and the pose information is extracted. Contrary to the original OpenPose and OpenPifPaf poses, we do not normalize the poses between 0 and 1 based on the bounding box dimensions, but between -1 and 1 based on the video frame dimensions. Our expectation is that the additional information on the pedestrian's position on the video frame might help the network to achieve better results. Furthermore, we believe that the normalization regarding the bounding box dimension is not ideal, as the bounding boxes' sizes vary significantly throughout an input sequence, which might prevent the network from learning meaningful features.

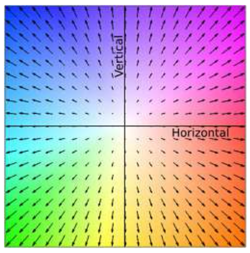
The resulting poses, including the confidence values for every keypoint, are then saved as Python pickle files. This enables us to run multiple tests without re-generating the poses every time.

Optical Flow

As stated above, optical flow is a popular technique used in different fields of behavior prediction. For our experiments, we decided to use FlowNet 2.0 to estimate the optical flow for every cropped pedestrian image, since our evaluated trajectory prediction network also uses FlowNet 2.0. The implementation we use is provided as a Docker image[8].

[8]https://github.com/lmb-freiburg/flownet2-docker


(a) Visualization of optical flow within pedestrian bounding boxes from the PIE dataset. For the visualization, flowiz was used.



(b) Color coding of the optical flow vector field. [MBW+20]

Figure 4.3: Optical flow visualization of pedestrians. The image on the right depicts the used color coding, where colors and saturation represent the flow vector direction and length, respectively.

To generate the optical flow images, we first save every cropped image from all the sequences generated by the PAB. The filenames are chosen in such a way, that every picture has its unique name based on the pedestrian ID. We further generate a Python pickle file, which contains all the sequences pointing to the corresponding pedestrian images. A custom script inside the docker environment then reads this pickle file and accesses the pedestrian images. To generate the optical flow, two images from a sequence, with a certain temporal distance, are necessary. The authors of [DFI+15] state that their algorithm works best for a video sequences with 5 frames per second. We therefore use a frame distance of 6, as the videos provided in the PIE dataset have a frame rate of 30 fps. One further advantage of the pickle file is that we can fetch the pedestrians' images from preceding sequences. Hence, there is only a gap of 6 frames for the first sequence. All subsequent sequences with the same pedestrian have optical flow data available at every timestamp. Finally, the optical flow data of every pedestrian image is converted to a PNG image using flowiz[9]. Examples of the converted pedestrian images are depicted in Figure 4.3a. Entire sequences of optical flow images used for training the network can be found in Appendix A.

To use the optical flow image as an input for SFRNN, we experiment with 3 approaches:

1. The optical flow images are directly passed to a standard VGG16 network, trained on the ImageNet dataset [DDS+09], to extract the features (as it is also done with the local context and the pedestrian image). The resulting feature vector is then passed to the SFRNN network instead of the poses. This approach is denoted as **SFRNN$_{OF}$**.
2. The STED (see Section 2.2.2) feature extractor is used to extract the features from the optical flow frame. The output vector is passed to the SFRNN network, replacing the pose input. This approach is denoted as **SFRNN$_{STED}$**

[9]https://github.com/georgegach/flowiz

3. Instead of using an external feature extractor, VGG16 is directly integrated into the SFRNN network such that it is trained with SFRNN to specialize on the task of pedestrian action prediction. We refer to this network as **SFRNN$_{VGG}$**.

Regarding the approach (1), the optical flow images are resized to 224×224 and then passed to a standard Keras pre-trained VGG16 model. The output of the network is then passed to the SFRNN network.

In approach (2), the STED feature extractor basically is the **ResNet** part inside the DTP network (see Figure 4.4). However, instead of using a ResNet18, a ResNet50 is used. In the following, we shortly summarize the changes made to the original ResNet50. First, the input and output layers are changed: The first convolutional layer uses $2m$ input channels (horizontal and vertical flow) rather than 3 (RGB), where m is the input sequence length. The other dimensions of the first layer are kept the same. The 1000-D softmax output layer is replaced with a 30-D fully connected layer which produces predictions for the x and y coordinates of the 15 future bounding box centroids. As the actual trajectory prediction output is not needed, the fully connected layer at the end of the network is replaced with a linear layer to pass through the features. This results in a 2048-dimensional feature vector which can then be used as an input for the SFRNN network. However, the ResNet50 network itself takes in a sequence of 9 optical flow images. Therefore, in order to experiment with the ResNet in our pipeline, a flow stack is created at every timestep for each sequence. Additionally, this flow stack needs to consist of flow images separated into horizontal and vertical flow images to meet the input requirements of the STED feature extractor. Hence, we convert the RGB flow images to grayscale horizontal and vertical optical flow images. The network is trained on the authors' [SGS20] custom subset of the JAAD dataset. Furthermore, cross-modality pre-training and partial batch normalization are applied as described in [WXW+16], to initialize the CNN with ImageNet weights. However, we argue that this flow stack input is not ideal for our use case, as SFRNN already processes a sequence. We hence feed redundant information to the network, as a flow stack with 9 optical flow images is generated for every time step of the sequence. To remedy this issue, we propose SFRNN$_{VGG}$ as a third approach.

In approach (3), using SFRNN$_{VGG}$, the SFRNN and the VGG networks are trained at the same time. This is achieved by replacing the GRU layer for the optical flow image input with the VGG16 network, wrapped in a TimeDistributed[10] layer. This layer allows us to train VGG16 with input sequences, since VGG16 by itself only expects a single image as an input. The TimeDistributed layer basically applies VGG16 on every temporal slice of the input sequence independently and then outputs the sequence of the predictions made by VGG16.

[10]https://keras.io/api/layers/recurrent_layers/time_distributed/

4.2.2 Changing Inputs of the Used Network

As already discussed in Section 2.2.1, SFRNN uses 5 inputs by default. The network architecture is depicted in Figure 4.1, which also shows the concatenation order of the input features. The concatenation is done as follows: For every input feature, there is a dedicated GRU. Each GRU takes one sequence of one feature as an input and outputs, in the case of SFRNN, a $(15, 256)$ feature vector. This is because 15 frames are used as input and the GRUs are initialized with an output dimensionality of 256. This resulting feature vector is then concatenated with the next feature sequence, yielding a feature vector of size $(15, 256 + x)$, where x is the length of the subsequent feature vector. The vector is then passed to the next GRU and the whole procedure is repeated for every feature and GRU. Towards the end of the network, the output from the last GRU is passed to a Fully Connected (FC) layer with a sigmoid activation function to get the actual crossing prediction.

When different inputs with different dimensionalities are used for our experiments, only the input dimensions of the corresponding GRUs needs to be changed. Everything else stays the same, meaning that the output dimension of that GRU and the concatenation procedure remain unchanged.

4.3 Trajectory Prediction

The goal of experiments on trajectory prediction is to evaluate, if a simple trajectory prediction network like DTP is able to produce satisfying results, which can then further be used for crossing prediction. DTP uses a stack of optical flow images to predict a person's trajectory 15 timesteps into the future. The network itself consists of a ResNet18 with the same modifications as the ResNet50 used for STED. These modifications were already discussed in Section 4.2.1. We will therefore focus on the input generation, output visualization and evaluation method.

Contrary to RNNs, CNNs do not expect a sequence of inputs, but only a single block of data. Therefore, DTP expects a stack of optical flow frames which consists of the 9 past frames before the time of prediction. The authors of DTP used FlowNet 2.0 to create the optical flow data of each pedestrian. The optical flow frames are then converted to grayscale images, containing the horizontal and vertical optical flow components. The stack is a 3-dimensional array of size $(256, 256, 9 \times 2)$, since the optical flow images are resized to 256×256 and 9 frames, each containing horizontal and vertical flow data, are used.

Using the flow stack as input, DTP outputs a vector containing 30 values, representing the correction values for the constant velocity assumption of the predicted trajectory (see Figure 4.4 for a visualization of the network). The first 15 values are the correction values for the x direction and the remaining 15 values are for the y direction. For our qualitative evaluation, visualizing the trajectories is necessary. Hence, we convert the correction values into 2D image space using the following procedure:

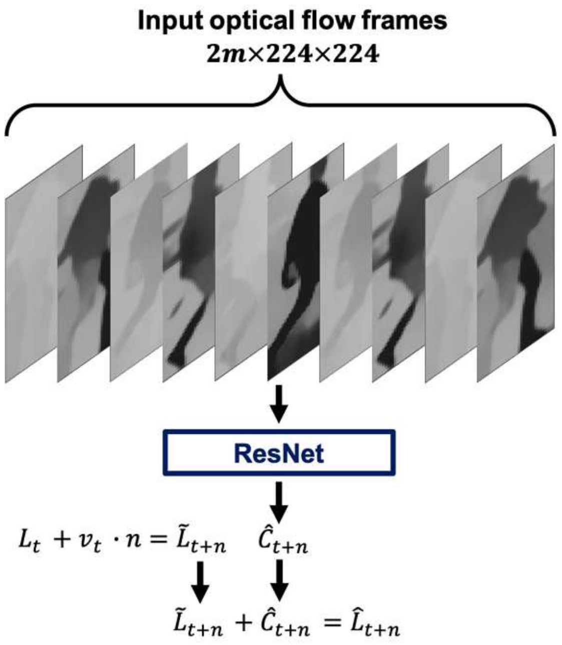

Input optical flow frames
2m×224×224
ResNet
$L_t + v_t \cdot n = \tilde{L}_{t+n}$
$\hat{C}_{t+n}$
$\tilde{L}_{t+n} + \hat{C}_{t+n} = \hat{L}_{t+n}$

Figure 4.4: DTP takes the last $2m$ optical flow frames as input and predicts pedestrian trajectories relative to a constant velocity baseline. The network is based on standard ResNet18, whereas the input and output layers are modified to match the desired feature vector sizes. [SRS19]

First, we apply the constant velocity assumption:

$$\tilde{L}_{t+n} = L_t + v_t \cdot n$$

where L_t denotes a pedestrian's centroid location at timestep t, v_t is the velocity at that timestep and n the number of future timesteps. These values need to be calculated based on the trajectory data given in the dataset. The authors of DTP provide a script which generates the values of the constant velocity assumption for each sequence in the JAAD dataset.

After having formulated the constant velocity assumption, we denote its error by:

$$\tilde{e}_{t+n} = L_{t+n} - \tilde{L}_{t+n}$$

where L_{t+n} is the actual trajectory. The compensation term is further defined as: $C_t = -\tilde{e}_t$. When predicting the compensation term, the actual trajectory prediction in the 2D image space $\hat{L}_{t+n}$ is calculated as follows:

$$\hat{L}_{t+n} = \tilde{L}_{t+n} + C_{t+n} \tag{4.1}$$

Using the converted 2D image space trajectory values, we can draw the predicted path on the input video (see Section 5.1).

For the quantitative evaluation, we use the Mean Squared Error (MSE) metric, which defines the average amount of deviation of the predicted trajectories from the ground truth trajectories. The MSE is calculated as follows:

$$MSE = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2 \tag{4.2}$$

where Y is the vector containing the ground truth data and $\hat{Y}$ the one containing the predicted values.

CHAPTER 5

Experiments and Results

In this chapter, we present the experiments and evaluations conducted on pedestrian detection and behavior prediction. As motivated in Chapter 3, we experiment with methods for trajectory prediction, person detection and action prediction. We evaluate the performance of DTP [SRS19] to assess if similar trajectory prediction methods based on optical flow may be used for crossing prediction or as an input for other action prediction networks. Next, the performance of Faster R-CNN [RHGS16] specifically on the task of pedestrian detection is evaluated: its performance is compared to the detection performance of OpenPifPaf [KBA21], which, being a bottom-up pose estimator, enables us to detect multiple persons and their poses on a single input image. Finally, we conduct a range of experiments with SFRNN [RKT20] which show how its performance can be improved using different input modalities including poses generated by different pose estimators and optical flow. Visualizations of input sequences used for training SFRNN can be found in Appendix A.

5.1 Trajectory Prediction

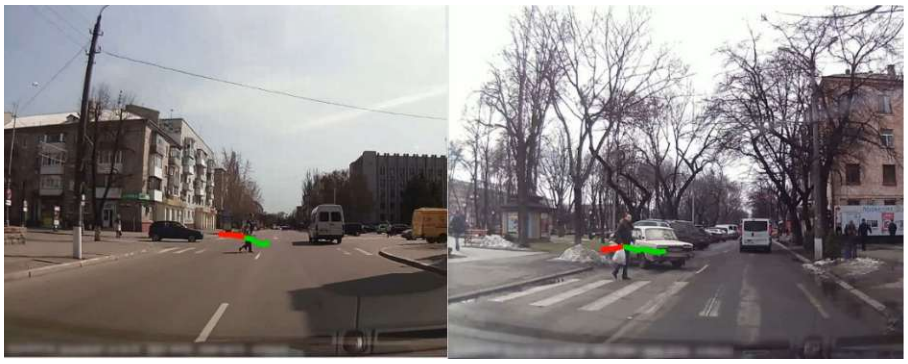
Trajectory prediction plays a role in risk potential estimation of pedestrians, as it can also be used for crossing prediction, which is necessary to prevent collisions. To estimate the applicability of trajectory prediction methods for crossing prediction, we evaluate DTP's (see Section 2.2.2) performance quantitatively and qualitatively using the JAAD dataset. The qualitative evaluations are conducted by visualizing the generated trajectories. Regarding the quantitative evaluations, we evaluate the performance of DTP between the vehicle states *moving* and *not moving*, to investigate how an optical flow based trajectory predictor is influenced by background motion. Based on the evaluation results, we consider using its output as input for SFRNN.

The DTP network takes a stack of optical flow images and outputs the predicted trajectory in the form of a vector with 30 correction values (correcting the values generated by the

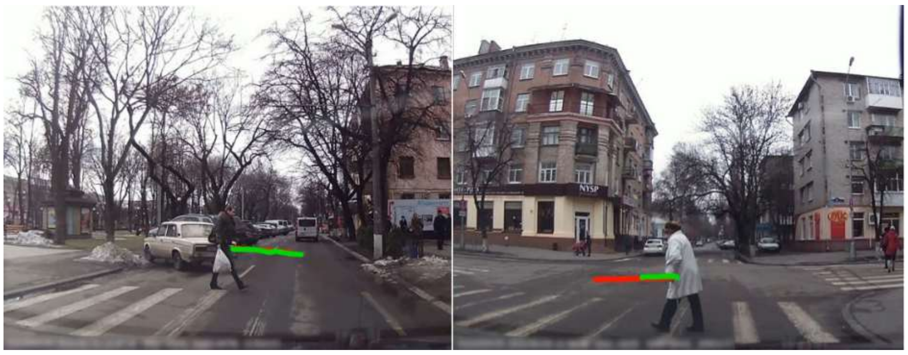
Driver behavioral tag	Stopped	Other than stopped
RMSE	21	25

Table 5.1: Evaluation of DTP on the JAAD dataset based on the driver's behavioral tag (moving slow, slow down, stopped), using the RMSE. The calculation is based on the absolute pixel values of images with the dimension of 1280×720.

linear velocity assumption), where the first 15 values are the correction values for the x direction and the remaining 15 values for the y direction. These correction values can be converted to the 2D image space as denoted in Equation 4.1. Example visualizations of the predicted trajectories are depicted in Figure 5.1. As the illustrations show, DTP seems to only generate satisfying trajectories when the car is stopped or moving slowly. Otherwise, the car's movement influences the predictions significantly. To confirm this assumption, we further quantitatively evaluate the performance of DTP specifically by calculating the Root Mean Squared Error (RMSE) when the car is stopped and when the car is driving. The RMSE is defined as $\sqrt{MSE}$, where the MSE is calculated as defined in Equation 4.2, using the absolute pixel values. The evaluation results are depicted in Table 5.1, showing that DTP indeed performs significantly worse when the car is moving. We draw the conclusion that a trajectory predictor solely working with optical flow images does not perform well enough, which also becomes evident when analyzing the video visualizations. Hence, we do not attempt to conduct further experiments on using the generated trajectories as inputs for SFRNN.



(a) Examples of DTP failing to predict correct trajectories: The car was turning towards the right during the observation timeframe. Therefore, the optical flow is altered in such a way that DTP predicts the trajectory in the opposite direction.



(b) Examples of DTP producing satisfying results: When the car is stopped or moving slowly, the trajectory's direction is almost identical to the GT trajectory.

Figure 5.1: Visualization of the predicted trajectories generated by DTP. Prediction is made one second into the future and footage from the JAAD dataset is used. The green trajectories are the GT trajectories, the red ones are the predicted trajectories.

5.2 Person Detection

The goal of the experiments on person detection is to evaluate if a bottom-up pose estimator can replace a dedicated object detector for generating pedestrians' bounding boxes. The bounding boxes are necessary, since many crossing prediction networks rely on different input modalities such as the image of the pedestrian or the pedestrian's surrounding, which can only be generated when the bounding box of the pedestrian is known. Moreover, the bounding boxes themselves are often used as an input for action prediction networks. For our experiments, Faster R-CNN and OpenPifPaf were chosen. While Faster R-CNN is a general object detector, OpenPifPaf is a bottom-up human pose estimator, which makes it also suitable for pedestrian detection. As poses are among the most common inputs for crossing prediction networks, we expect this experiment to provide valuable information on the possibility of improving the generation of bounding boxes by only using one network which jointly produces the pedestrian poses

Metric	Area Size	FasterRCNN$_{ped}$	FasterRCNN$_{all}$	OpenPifPaf
$AP^{0.50:0.95}$	all	**0.37**	0.31	0.08
$AP^{0.50}$	all	**0.73**	0.66	0.22
$AP^{0.75}$	all	**0.32**	0.26	0.04
$AP^{0.50:0.95}$	small	**0.20**	0.16	0.01
$AP^{0.50:0.95}$	medium	**0.47**	0.42	0.13
$AP^{0.50:0.95}$	large	**0.65**	0.58	0.32
$AR^{0.50:0.95}$	all	**0.13**	0.12	0.05
$AR^{0.50:0.95}$	all	**0.40**	0.36	0.10
$AR^{0.50:0.95}$	all	**0.47**	0.40	0.10
$AR^{0.50:0.95}$	small	**0.36**	0.27	0.01
$AR^{0.50:0.95}$	medium	**0.56**	0.52	0.17
$AR^{0.50:0.95}$	large	**0.70**	0.64	0.38

Table 5.2: Evaluation results of Faster R-CNN and OpenPifPaf on the task of person detection. The subset BDD_{ped} contains only pedestrian samples. BDD_{all} contains all classes present in the BDD100K dataset. Faster R-CNN was trained on the two subsets of the BDD100K dataset, whereas a pre-trained model of OpenPifPaf (trained on the COCO dataset) was used. The models are evaluated on a subset of the BDD100K testset containing only pedestrian samples, as we are solely interested in the pedestrian detection performance. FasterRCNN$_{ped}$ and FasterRCNN$_{all}$ denote Faster R-CNN trained on BDD_{ped} and BDD_{all}, respectively. Evaluation metrics are based on the COCO object detection challenge.

and bounding boxes.

As the PIE dataset does not contain annotations for every pedestrian, we train Faster R-CNN with the BDD100K dataset. Since we are only interested in the detection of pedestrians, we also conduct experiments on how the performance of Faster R-CNN is altered, if we only train it with the pedestrian instances in comparison to training it with all classes present in the BDD100K dataset. The evaluation results after training the network for 10 epochs are depicted in Table 5.2, whereby the metrics from the COCO object detection challenge were used for the evaluation (see Section 3.4.2 for a detailed explanation). Our experiments show that Faster R-CNN performs significantly better for pedestrian detection if it is only trained on pedestrian samples. We hypothesize that the network is able to detect pedestrians more easily, as the region proposal network only focuses on one label type. This may simplify the bounding box regression, which consequently does not need to differentiate between different types of bounding boxes.

Regarding OpenPifPaf, the best performing pre-trained model (trained on COCO) provided on the OpenPifPaf website[1] is used. The evaluation shows that it performs clearly worse on the task of object detection in comparison to Faster R-CNN. OpenPifPaf especially struggles with small pedestrians: In relative comparison to Faster R-CNN, the average recall for large bounding boxes drops by 46% (0.7 to 0.38) if OpenPifPaf is used. For small bounding boxes, the average recall even decreases by 97% (0.36 to 0.01), which indicates that OpenPifPaf in essence is not able to detect small pedestrians.

5.3 Pedestrian Action Prediction

The goal of our experiments on pedestrian action prediction is to increase the performance of the evaluated crossing prediction network. We hypothesize that changing the inputs of our evaluated network may improve the network's performance and therefore experiment with different input modalities and evaluate which of them yield the best results.

Since SFRNN achieves state-of-the-art results in the PAB [KRT21], we chose it as a framework for evaluation in our experiments on predicting pedestrians' crossing intentions. We found the poses to be unreliable input, as experiments conducted on the JAAD and PIE dataset showed OpenPose to generate unsatisfactory pose estimations for small images of pedestrians (see Figure 5.2 for visualizations of poses generated by OpenPose). This is particularly problematic, since pedestrians usually appear small in datasets for autonomous driving, where road scenes with many road users are depicted. In the following, we refer to these pedestrian samples as "small pedestrians". Therefore, we experiment with different pose estimators to increase the poses' quality, which potentially improves the overall performance of SFRNN. Furthermore, we experiment with replacing the poses by optical flow, since optical flow is a popular feature for behavior prediction (as stated in Section 4.2.1). We hypothesize that optical flow is more reliable than pose estimation and will result in better action prediction performance.

5.3.1 Analyzing and Improving Pose Inputs

In the PAB, Kotseruba et al. [KRT21] provide pre-generated poses for the JAAD and PIE datasets using OpenPose. After an initial qualitative analysis of the data, we found that many pedestrian samples have no poses assigned or contain severely misplaced or missing keypoints. We therefore examined how many pedestrian bounding boxes actually have a corresponding pose and calculated the percentage of available poses per dataset. Furthermore, we compared these results with the number of detected poses of OpenPifPaf as it also performs better on the COCO dataset in comparison to OpenPose [KBA21]. A quantitative comparison between OpenPose and OpenPifPaf can be found in Table 5.3. It is worth noting, however, that this metric only gives a rough idea of the detection performance, since it still does not show the level of quality of the predicted poses. Additionally, there are many bounding boxes in the datasets, where the pedestrian is occluded. This fact is not taken into account by this metric.

[1]`https://openpifpaf.github.io/intro.html#pre-trained-models`

On average, only 43% of the pedestrian bounding boxes within the JAAD and PIE datasets have a pose generated by OpenPose. With OpenPifPaf, the average percentage of available poses is slightly higher with 47%. While training SFRNN, in case of a non-existent pose, the pose input vector only contains zeros. We therefore experimented with different pose estimation networks to evaluate, if the number of detected poses may be increased and may consequently result in better intent prediction performance. Furthermore, we analyzed how much impact the absence of poses has on the network for each dataset. This is done by setting all values in the pose input vector to zeros. The evaluation results of SFRNN trained with these zero pose vectors can be found in Table 5.5. Interestingly, regarding the PIE dataset, the poses have no noticeable influence on the prediction quality. However, for the JAAD$_{all}$ dataset, the difference is significant and the OpenPose poses (although with unsatisfying quality) show to improve the prediction quality. Regarding the JAAD$_{beh}$ dataset, the situation is similar to the PIE dataset, whereas the OpenPose poses even slightly degrade the results in comparison to the zero pose vectors. This might be related to the poor quality of poses used, as further discussed in this section.

To further consolidate our hypothesis that the number of zero vector poses is related to the overall performance of SFRNN, we replace the sampling method for cropping the pedestrian images. The PAB uses the nearest neighbor method to sample the cropped images, which results in pixelated images. We hence use the Lanczos filter instead. Results showing the percentages of detected poses by OpenPifPaf, with the two different sampling methods, are depicted in Table 5.4. As the results show, OpenPifPaf detects poses for on average 73% of bounding boxes, which is a 30% improvement to the pose estimation method provided by the authors of SFRNN (43% with OpenPose applied on bounding boxes resampled with nearest neighbor method). We therefore use the Lanczos resampling filter from now on for our experiments with pose estimation networks.

The evaluation of SFRNN using OpenPifPaf is depicted in Table 5.5. Interestingly, the evaluation shows that OpenPifPaf performs slightly worse on all datasets, while the differences are more significant on the JAAD$_{all}$ and JAAD$_{beh}$ dataset. The OpenPifPaf poses seem to degrade the performance of SFRNN, although the number of detected poses is significantly higher in comparison to OpenPose. However, with the most challenging

Dataset	OpenPose	OpenPifPaf
PIE	42.49%	**49.58%**
JAAD$_{all}$	26.97%	**30.21%**
JAAD$_{beh}$	59.53%	**62.23%**
Average	43.00%	**47.00%**

Table 5.3: Percentage of pedestrian bounding boxes with poses. The JAAD$_{all}$ dataset has the lowest percentage of available poses.

Dataset	Images cropped with NN	Images cropped with Lanczos
PIE	49.58%	**77.77%**
$JAAD_{all}$	30.21%	**59.09%**
$JAAD_{beh}$	62.23%	**82.02%**

Table 5.4: Percentage of pedestrian bounding boxes with poses detected by OpenPifPaf. Pose estimation applied on images cropped with the Lanczos filter has a higher success rate as opposed to applying pose estimation on images cropped with the nearest neighbor (NN) method.

dataset, JAAD$_{beh}$, the recall is higher with the OpenPifPaf poses.

To further improve the performance of SFRNN, we add the confidence value for each detected keypoint to the pose vector. We hypothesize that the network may learn to better comprehend the pose inputs by giving less weight to poses with low confidence values, as they are less reliable. By doing so, the size of the pose input vector is increased from 34 (17 keypoints with x and y coordinates) to 51 values , i.e., every keypoint with a x and y coordinate now also has a confidence value in the range from 0 to 1. In Table 5.5, SFRNN trained with the OpenPifPaf poses including the confidence values is denoted as OpenPifPaf$_{conf}$. The results show that SFRNN trained with the OpenPifPaf$_{Conf}$ poses achieves higher accuracy and recall values in comparison to SFRNN trained with the standard OpenPose poses. The increase in recall on the PIE and JAAD$_{beh}$ dataset is particularly significant.

We further experiment with the baseline method Simple Pose [XWW18] provided in the GluonCV toolkit[2]. Pre-trained models are available, which surpass the AP of OpenPifPaf by 3.9% on the 2017 COCO challenge test set (from 70.9% to 74.8% given the highest performing Simple Pose model provided by GluonCV).

As higher numbers of available poses (i.e., lower number of zero pose vectors) showed to overall improve the performance of SFRNN, we go one step further and change the confidence threshold of Simple Pose to alter the number of zero vector poses. Simple Pose is a top-down approach, meaning that a cropped image of the already detected person is passed to the network. Hence, the network always outputs keypoints, even if no person is present. Of course, in that case, the confidence values for the keypoints are low. Therefore, the threshold values 0.0, 0.2, 0.4 and 0.7 were tested, where the number of zero vector poses with the 0.7 threshold was the same as with OpenPifPaf, which uses 0.2 as its threshold. Simple Pose detects on average poses for 98.51% on all bounding boxes of the JAAD and PIE dataset with the standard 0.2 threshold (26% higher in comparison to OpenPifPaf). This indicates that Simple Pose has a much higher recall in comparison to OpenPifPaf when it comes to keypoints detection.

[2]https://cv.gluon.ai/build/examples_pose/demo_simple_pose.html

We trained SFRNN with the Simple Pose poses and the different thresholds and found the poses with the 0.0 confidence threshold (i.e., every pose input vector contains values, there are no zero poses) to work best on average over the PIE and JAAD dataset. Moreover, as the confidence values output by the pose estimation network are now used for SFRNN, it makes sense to not filter the keypoints, since the network should learn to interpret the confidence values by itself. In Table 5.5, we therefore depict the training results of SFRNN trained with Simple Pose poses with the confidence threshold set to 0.0. Interestingly, SFRNN trained with the Simple Pose poses performs worse on the PIE and JAAD$_{beh}$ datasets in comparison to the model trained with the OpenPifPaf poses. However, the performance is significantly better on the JAAD$_{all}$ dataset.

Finally, we experiment with using HRNet for the pose generation. We qualitatively evaluate the visualizations of the pose estimations and find the performance of HRNet trained on the BDD100K dataset satisfying. A comparison of pose estimations between the original OpenPose poses and the poses generated by HRNet is depicted in Figure 5.2 (for visualizations of entire sequences of pose estimations used for training the network see Appendix A). While OpenPose fails to detect certain limbs (Figures 5.2a and 5.2b), or even fails to detect any limb for particularly challenging images (Figures 5.2c and 5.2d), HRNet is able to produce satisfactory results. Furthermore, the problem of using a bottom-up pose estimator becomes evident: In Figure 5.2e, OpenPose detects two or more persons. Since in the generation process, always the first pose has been selected from the OpenPose output, the wrong pose could have been selected occasionally, like in this case. HRNet, being a top-down pose estimator, using the ground truth bounding boxes, does not suffer from this problem.

As the qualitative evaluation of the HRNet output shows to produce significantly better results in comparison to OpenPose, we train SFRNN without using the confidence values of the pose estimation. Furthermore, we find the network to converge slowly during training for the JAAD datasets. We therefore change the learning rate from 10^{-7} to 5×10^{-4}, which showed to significantly improve the performance on these datasets. The final evaluation results of SFRNN trained with optimized learning rate are depicted in Table 5.5.

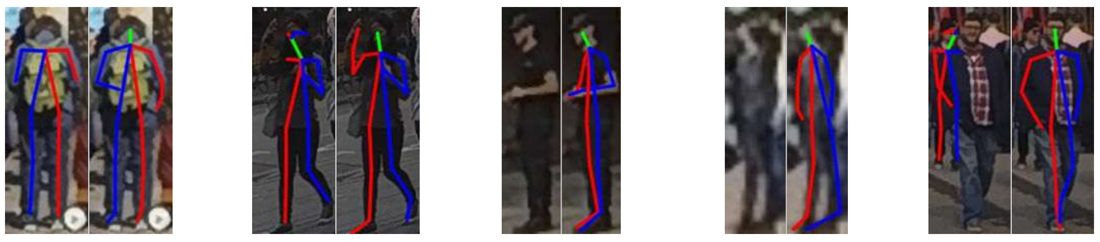

(a) (b) (c) (d) (e)

Figure 5.2: Comparison of poses generated by OpenPose and HRNet: For each image pair, the left image depicts the pose generated by OpenPose and the right image the pose generated by HRNet.

Dataset	SFRNN using poses from	acc	auc	f1	precision	recall
PIE	OpenPose (original)	0.82	0.78	0.69	0.70	0.70
	Zero Vector Poses	0.83	0.78	0.69	0.70	0.68
	OpenPifPaf	0.82	0.76	0.67	0.70	0.63
	OpenPifPaf$_{Conf}$	0.83	0.84	0.74	0.64	**0.86**
	Simple Pose	0.83	0.77	0.68	0.72	0.64
	HRNet	**0.87**	**0.87**	**0.79**	**0.73**	0.85
JAAD$_{all}$	OpenPose (original)	0.72	0.71	0.47	0.35	0.70
	Zero Vector Poses	0.68	0.66	0.40	0.30	0.63
	OpenPifPaf	0.67	0.68	0.42	0.30	0.68
	OpenPifPaf$_{Conf}$	0.73	0.74	0.49	0.36	0.75
	Simple Pose	0.85	0.81	0.64	0.56	0.75
	HRNet	**0.89**	**0.87**	**0.75**	**0.66**	**0.87**
JAAD$_{beh}$	OpenPose (original)	0.55	0.58	0.55	**0.72**	0.45
	Zero Vector Poses	0.57	0.58	0.60	0.71	0.51
	OpenPifPaf	0.51	0.49	0.60	0.62	0.58
	OpenPifPaf$_{Conf}$	0.58	**0.59**	0.62	**0.72**	0.55
	Simple Pose	0.56	0.56	0.62	0.68	0.57
	HRNet	**0.63**	0.53	**0.76**	0.64	**0.93**

Table 5.5: Evaluation results of SFRNN trained with inputs from different pose estimation networks (except *Zero Vector Poses*, which indicates, that vectors, only containing zeros were used as pose inputs, i.e., no pose estimation network was used).

As the results shows, SFRNN trained with the HRNet poses clearly outperforms the original SFRNN results using OpenPose poses. It also outperforms the pose estimator alternatives Simple Pose and OpenPifPaf.

5.3.2 Replacing Poses with Optical Flow

Optical flow is a popular input feature in the field of behavior prediction (see Section 4.2.1). We therefore conduct experiments on replacing the pose inputs with optical flow and evaluate, if the performance of SFRNN is increased.

The training results of the different SFRNN networks using optical flow are depicted in Table 5.6. SFRNN$_{OF}$ uses a standard VGG16 network, trained on the ImageNet dataset [DDS$^+$09], to extract the features from the RGB optical flow images. The performance of SFRNN$_{OF}$ is similar on the PIE dataset compared to the SFRNN version using the HRNet poses. However, the performance dropped considerably on both JAAD dataset versions.

As the experiments show, optical flow has the potential to further increase the performance of our action prediction network. Nevertheless, we found that in certain situations it may also degrade the results, as the optical flow images are processed with a standard VGG16 network, which was trained on ImageNet for object recognition purposes. We

Dataset	Network	acc	auc	f1	precision	recall
PIE	SFRNN$_{OF}$	**0.88**	**0.87**	**0.79**	**0.75**	**0.84**
	SFRNN$_{STED}$	0.82	0.77	0.67	0.68	0.66
	SFRNN$_{VGG}$	0.84	0.83	0.74	0.69	0.79
	SFRNN$_{VGG-A}$	0.81	0.76	0.66	0.66	0.66
JAAD$_{all}$	SFRNN$_{OF}$	0.76	0.62	0.37	0.35	0.40
	SFRNN$_{STED}$	0.68	0.70	0.44	0.32	**0.72**
	SFRNN$_{VGG}$	**0.85**	**0.79**	**0.62**	**0.56**	0.69
	SFRNN$_{VGG-A}$	0.71	0.67	0.42	0.33	0.61
JAAD$_{beh}$	SFRNN$_{OF}$	0.46	0.52	0.39	**0.66**	0.28
	SFRNN$_{STED}$	0.52	0.54	0.55	**0.66**	0.47
	SFRNN$_{VGG}$	**0.66**	**0.57**	**0.77**	**0.66**	**0.93**
	SFRNN$_{VGG-A}$	0.48	0.44	0.59	0.58	0.59

Table 5.6: Evaluation results of different customized versions of SFRNN using optical flow as an input: SFRNN$_{OF}$ (SFRNN using optical flow, processed by a pre-trained VGG16, instead of poses), SFRNN$_{STED}$ (SFRNN using the features extracted by STED feature extractor instead of poses), SFRNN$_{VGG}$ (SFRNN with integrated VGG16, which is trained at the same time), SFRNN$_{VGG-A}$ (the same as SFRNN$_{VGG}$, but using all 6 inputs instead of 5)

hypothesize that the results may be better if a network specialized on feature extraction of optical flow images is used. Hence, inspired by the STED network [SGS20], we use a custom ResNet50 for extracting features from the flow images.

The results of SFRNN$_{STED}$ using the STED feature extractor, which is specialized on optical flow images, showed to actually decrease the performance on the PIE dataset (see Table 5.6). However, regarding the JAAD datasets, the performance increased significantly in comparison to SFRNN$_{OF}$, particularly the recall. This shows that a feature backbone can improve the performance, if it is specialized on the specific task. With SFRNN$_{VGG}$, the VGG16 network is integrated into the SFRNN network. This means that the networks share the same weights and are therefore trained at the same time. As the evaluation shows, our SFRNN$_{VGG}$ overall achieves the best results over all SFRNN network versions using optical flow. Since the pose inputs generated by OpenPifPaf have shown to also improve the performance of the original SFRNN network, we expand our SFRNN$_{VGG}$ to also use the pose inputs. Hence, the network now uses 6 inputs in total: Pedestrian appearance, local surrounding, vehicle speed, pose, optical flow and the bounding boxes. We refer to this network as SFRNN$_{VGG-A}$.

As the evaluation results show, the additional input heavily degrades the performance of the network. We hypothesize that the datasets do not contain enough samples (i.e., are not large enough) to train a network with that amount of parameters. Recent work on this topic even suggests that bounding boxes are the only input needed for a crossing prediction network [AMF^{+}21]. However, in that work the tests were solely conducted on the PIE dataset. We also conducted experiments with the standard SFRNN and found, that only using bounding boxes as input indeed yields good results on the PIE dataset. Nevertheless, the results on the JAAD dataset are significantly worse compared to the SFRNN using all inputs. We found the performance of certain inputs to heavily rely on the used dataset. We therefore argue that the usage of multiple inputs is still beneficial for the performance of the network, especially if it is used in different settings.

CHAPTER 6

Conclusion and Future Work

In this thesis, we have shown the capabilities and problems of current state-of-the-art pedestrian crossing prediction methods. Such techniques are crucial for an autonomous vehicle to make safe decisions. To estimate the risk potential of a vulnerable road user, the predicted action and/or the predicted trajectory may be used.

We first evaluated the feasibility of generating bounding boxes of pedestrians by training Faster R-CNN on the BDD100K dataset. We showed that reducing the dataset to only one class (the pedestrian class) significantly improves the performance of Faster R-CNN when only one class needs to be detected. Furthermore, we compared the performance of Faster R-CNN to OpenPifPaf for pedestrian detection, which, especially for pedestrians seen from larger distances, showed to perform worse. Hence, we concluded that a bottom-up pose estimation network cannot replace a dedicated object detector without special modification or fine-tuning. Next, we experimented with the trajectory prediction network DTP and visualized its output to evaluate the performance qualitatively. The evaluation showed that in application to the JAAD and PIE datasets, a simple trajectory predictor like DTP does not produce adequate results and therefore, we did not conduct any further experiments on using its outputs for crossing prediction. As one of our core experiments, we evaluated the performance of SFRNN using different inputs, which showed that the type and quality of inputs is important. We found that the accuracy of detected joint positions strongly influences the evaluation results. In our experiments, the network trained with the poses generated by HRNet, which was originally trained on the BDD100K dataset, performed best. Besides our work on pose estimators, we trained our own version of SFRNN, which does not use poses but optical flow images generated from the cropped pedestrian images. This network includes a VGG16 network for the interpretation of the optical flow images, which were generated by FlowNet 2.0 beforehand. We named this version SFRNN$_{VGG}$. This network performed well, increasing the average accuracy over all datasets by 8 percentage points in comparison to the original SFRNN implementation. However, it did not exceed the results of SFRNN using the

HRNet poses, which on average achieved a 10 percentage points better accuracy over all datasets. This indicates that poses are indeed a good feature if they meet a certain level of quality. The experiments on optical flow showed that the flow images also are a reliable feature to use. However, considering the computational cost of training a network like SFRNN$_{VGG}$, generating and passing the less complex pose features to the network is preferred. In a final experiment, we combined our approaches for pose estimation and optical flow generation, which means that 6 inputs were used in total. However, this network performed worse. We hypothesize that more inputs confuse the network, as not enough training data is available for such a configuration.

To conclude, our results show that the performance of pedestrian crossing prediction networks significantly fluctuate between different datasets. Regarding SFRNN, we found that some inputs may work well for one dataset, but decrease the performance for another. In particular, with the PIE dataset, SFRNN performs best with simple inputs like poses and bounding boxes. However, with the JAAD dataset, further, more complex inputs like optical flow showed to clearly improve the performance, but slightly decreased the performance on the PIE dataset. Additional experiments with bigger datasets are necessary to conclude which inputs would overall perform best. As our experiments have shown, the reliability for pedestrian crossing and trajectory prediction of the networks we evaluated, is certainly not high enough to be used for fully autonomous driving. Especially the generation of multiple inputs used for pedestrian crossing prediction, which again relies on other networks, can be problematic. However, in the context of driver assistance, currently available systems may provide a valuable contribution to road safety.

6.1 Future Work

Many advances in autonomous driving have been made in recent years. However, as our experiments have shown, the systems which are currently available are not yet ready for full autonomous driving.

Pedestrian crossing prediction is just a small part of an autonomous vehicle: Systems for autonomous driving consist of many different neural networks which need to be trained separately and function jointly. This results in very complex systems which may fail unexpectedly. Therefore, further improvements in network architectures and training procedures need to be made to increase the reliability of these systems. Our experiments showed that different input types with different quality significantly influence the performance of crossing prediction networks, which again may perform differently on different datasets. Another promising approach would be to create an end-to-end trainable multi-task neural network, which is able to extract and predict multiple features as opposed to the currently used "multiple networks" architectures. We hypothesize that such a network may be able to perform better on the specific task of crossing prediction in the context of autonomous driving, as the network would be able to specialize on this particular type of data.

Thinking more long-term, autonomous cars may be able to detect and predict the behavior of traffic actors with a single multi-task network just by using a RGB camera input. This would most closely resemble the way a human would recognize and estimate the intent of other traffic actors just by using his or her eyes. All in all, we expect that more exciting research related to behavior prediction in the context of autonomous driving will take place in the years to come.

APPENDIX A

Visualizations of Inputs

This Appendix contains visualizations of the inputs used for training our evaluated pedestrian crossing prediction network. All samples are taken from the PIE dataset. As discussed in Section 4.2, our evaluated network uses five inputs: The vehicle's speed, the pedestrian's bounding boxes, poses, local context and appearance. In order to keep the overview compact, the vehicle's speed is omitted. Furthermore, the local context, appearance and bounding box of the pedestrian are combined. We therefore visualize the following inputs:

a) Local context with pedestrian appearance and bounding box.

b) Optical flow within the pedestrian bounding box generated with FlowNet 2.0 (Section 2.4). The color coding used for visualizing the optical flow is depicted in Figure 4.3b.

c) Pedestrian's poses generated with OpenPose (provided in the PAB). Limbs on the right, left and the middle part of the body are depicted in red, blue and green, respectively.

d) Pedestrian's poses generated with HRNet. The color coding of these poses is the same as with the OpenPose poses.

The samples are divided in crossing and non-crossing samples. The network uses a sequence of 15 frames to predict the crossing action one second into the future. Therefore, for each input feature, 15 frames are depicted.

A.1 Crossing Samples

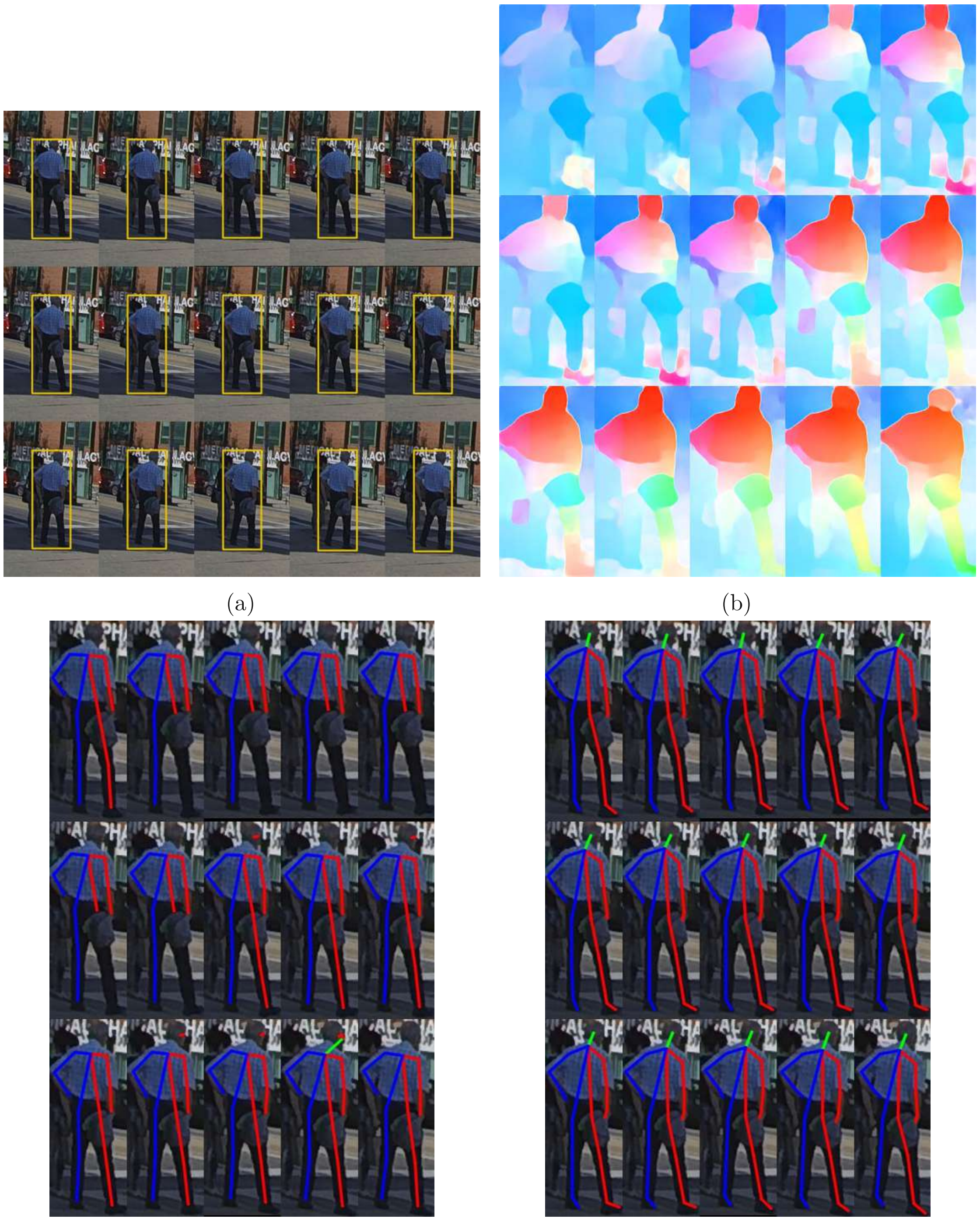
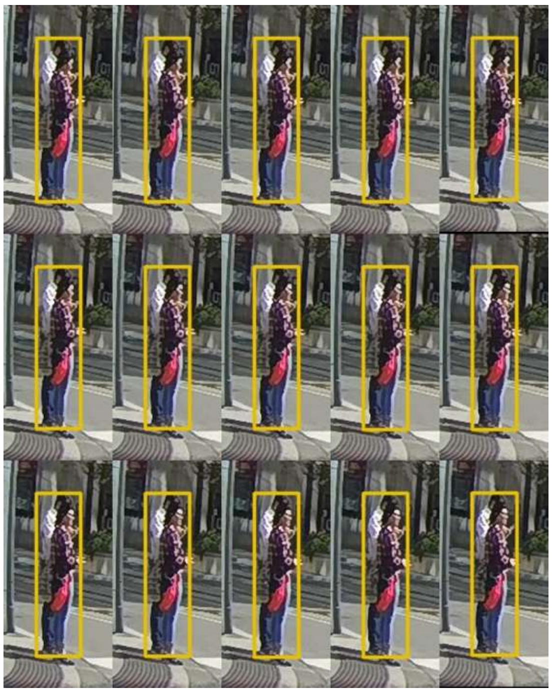
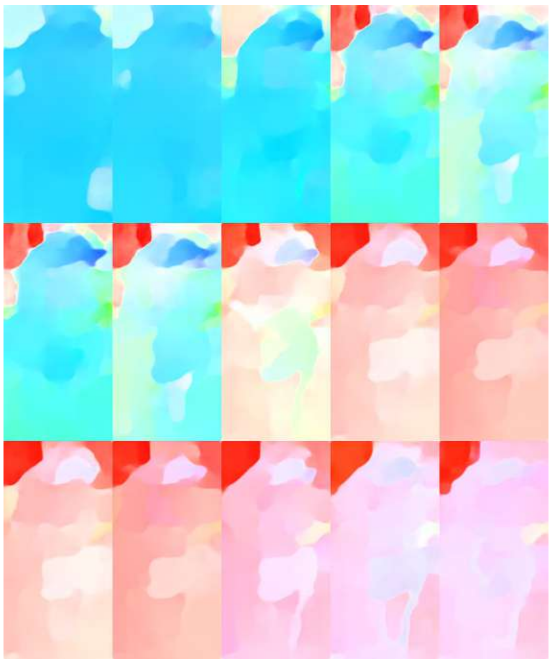
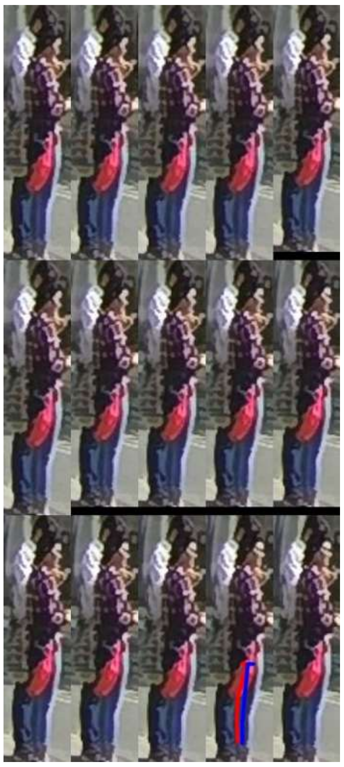


Figure A.1: Example crossing sequence 1.


(a)


(b)


(c)

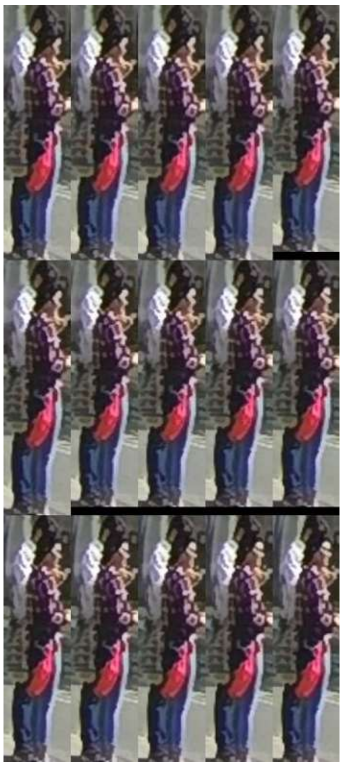
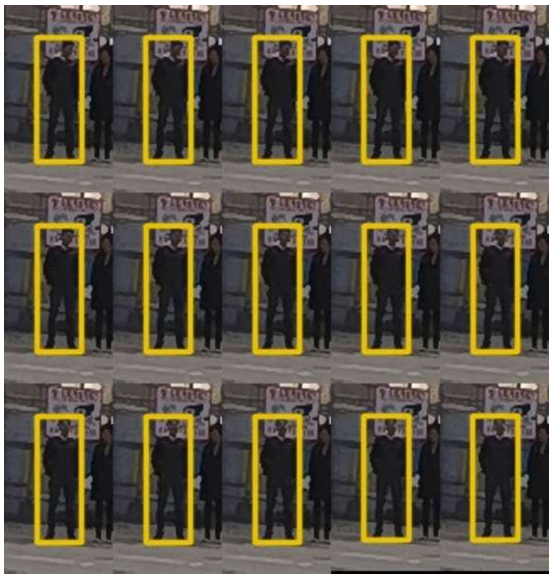
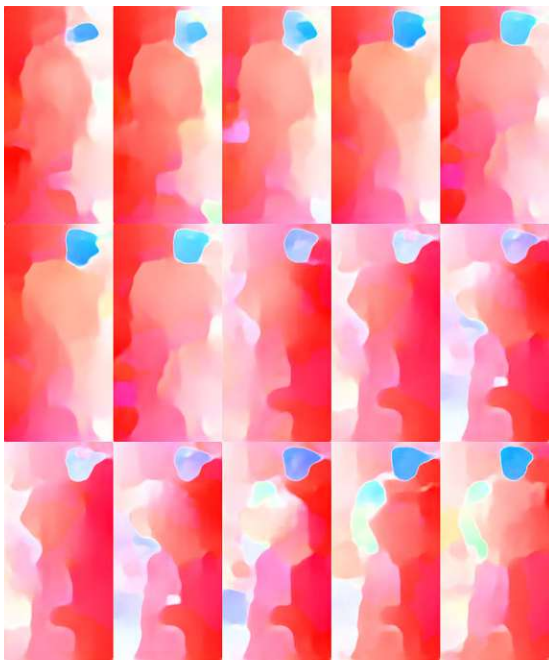
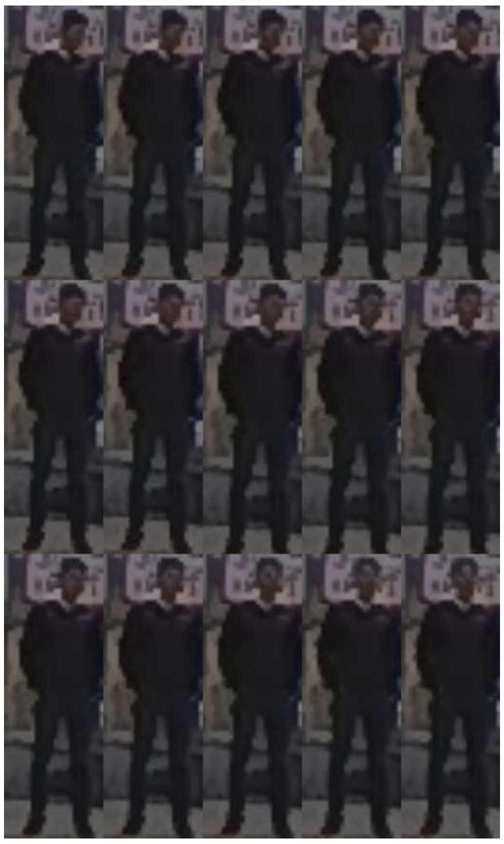

(d)

Figure A.2: Example crossing sequence 2.


(a)


(b)


(c)

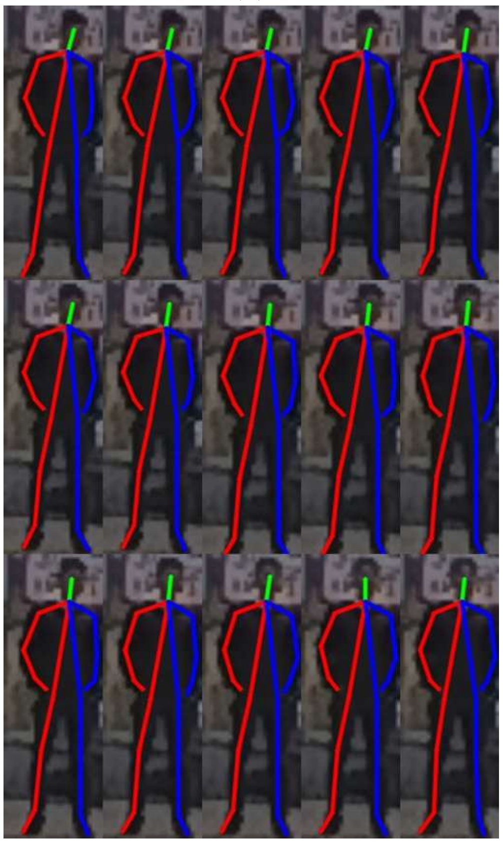
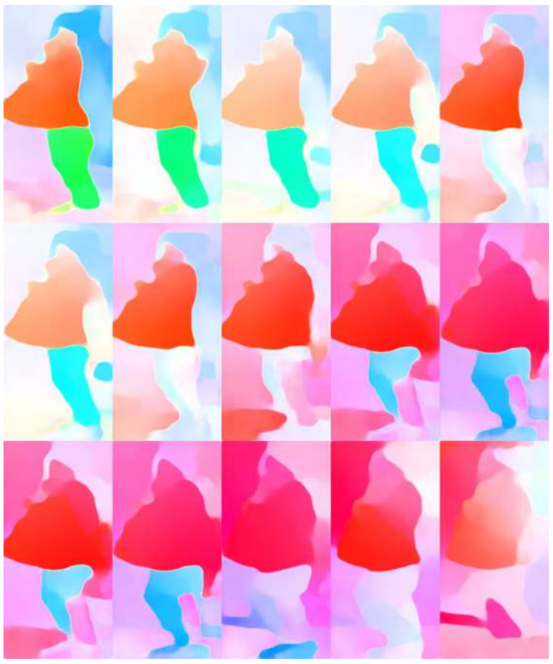
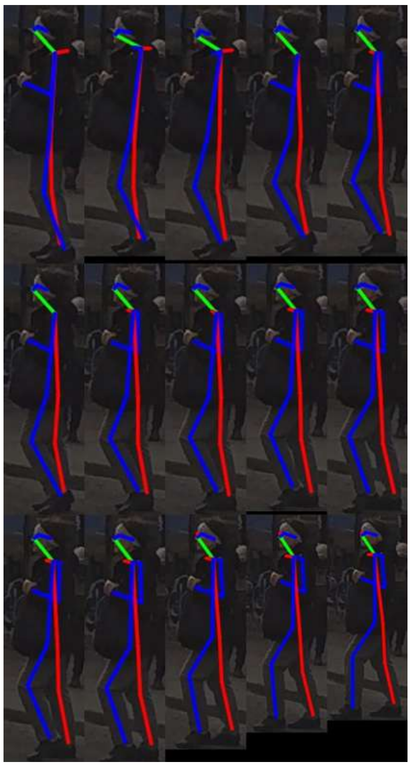

(d)

Figure A.3: Example crossing sequence 3.

(a)


(b)


(c)

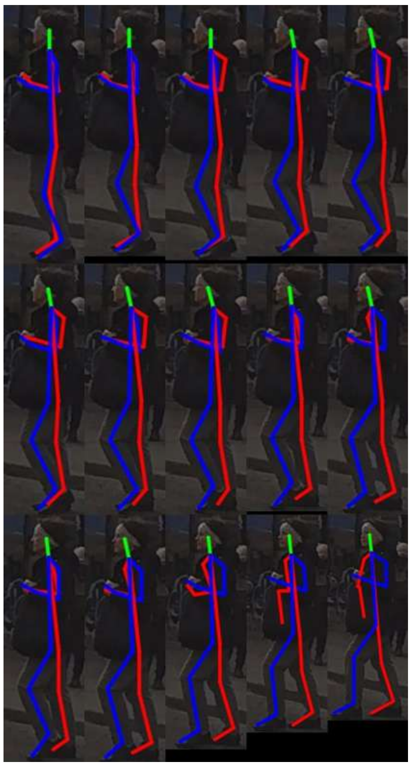
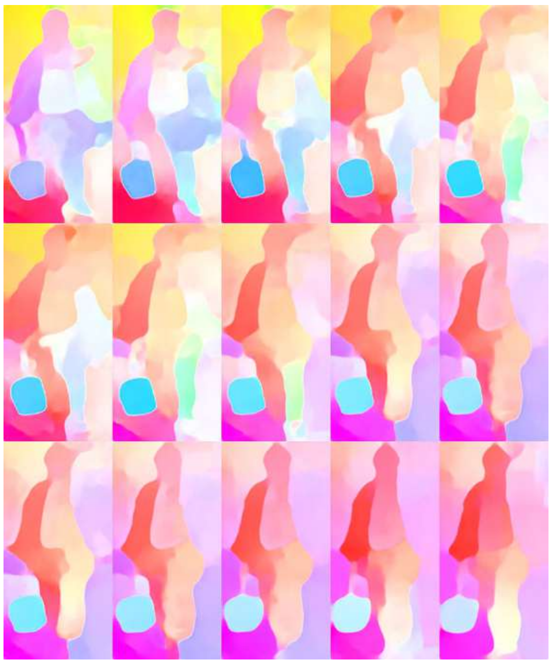
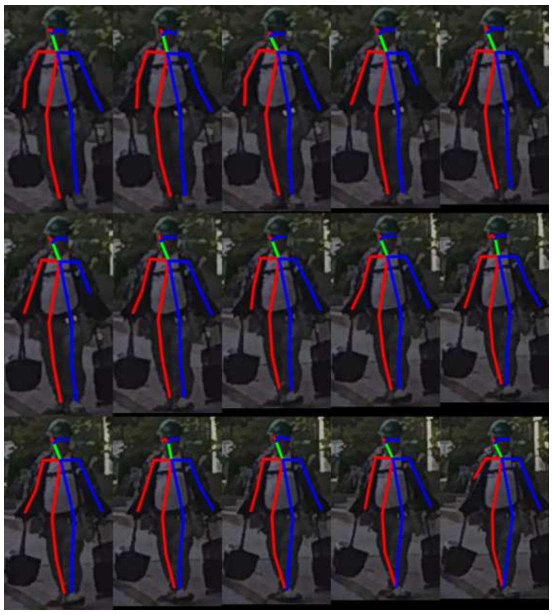

(d)

Figure A.4: Example crossing sequence 4.

(a)


(b)


(c)

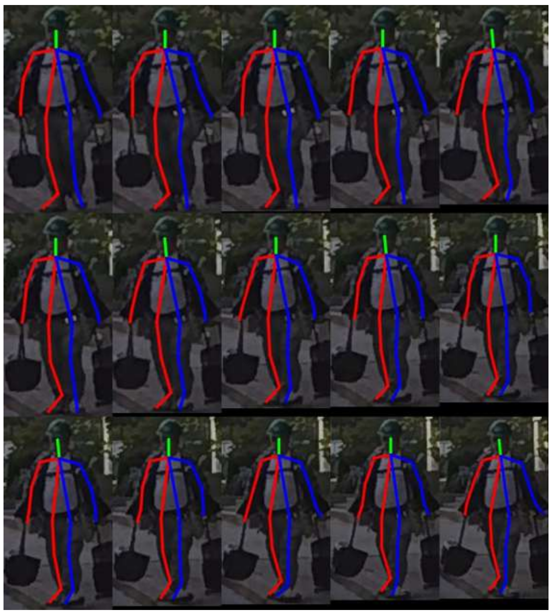
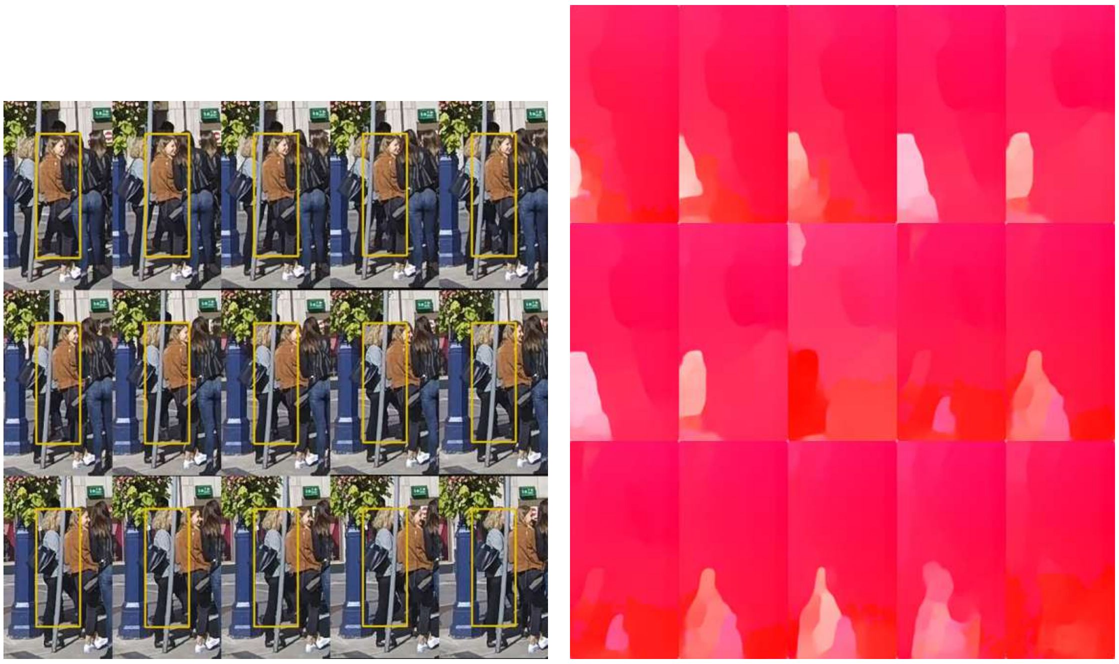

(d)

Figure A.5: Example crossing sequence 5.


(a) (b)

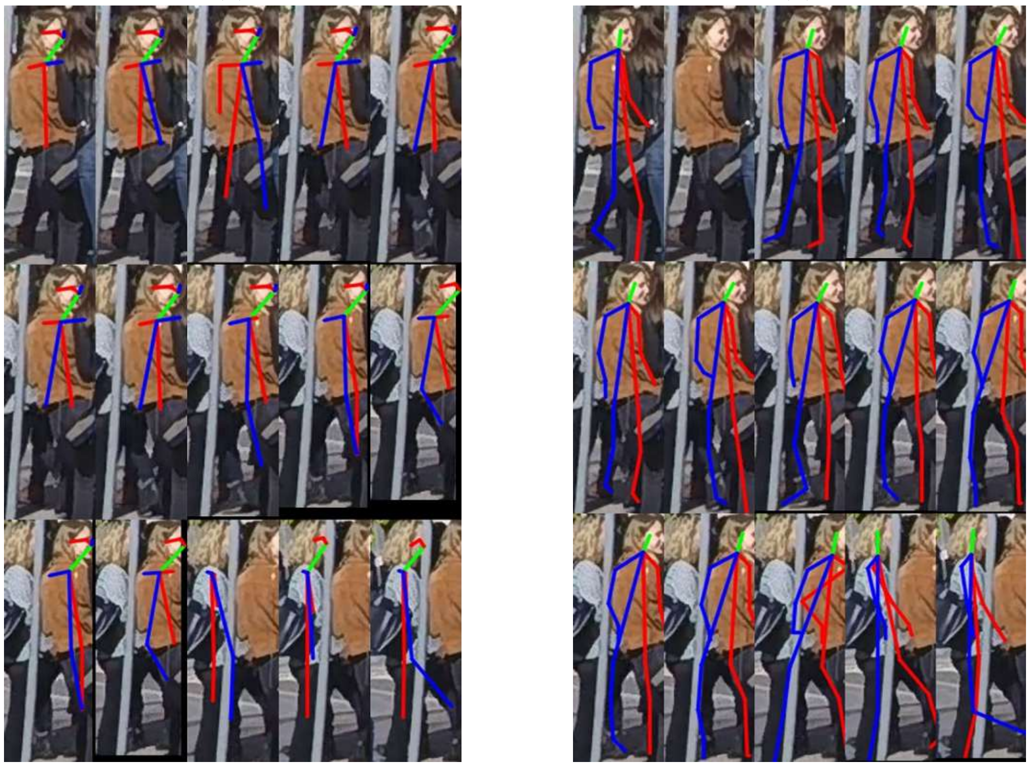
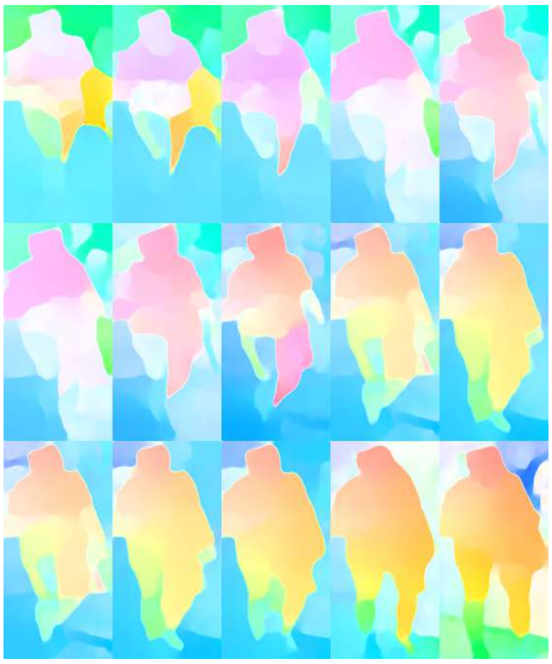
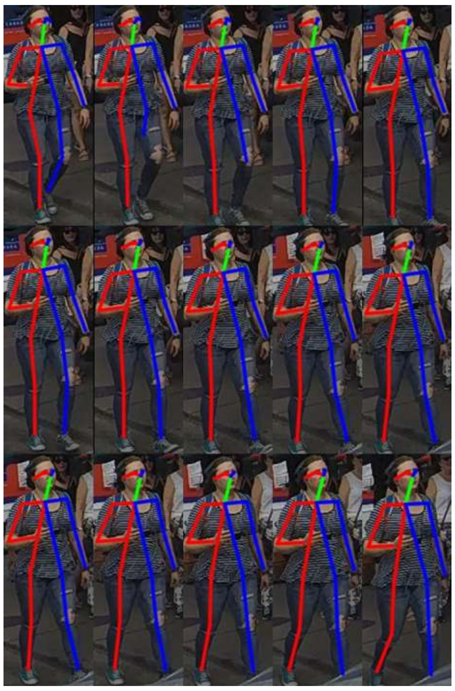

(c) (d)

Figure A.6: Example crossing sequence 6.

A.2 Non-crossing Samples

(a)


(b)


(c)

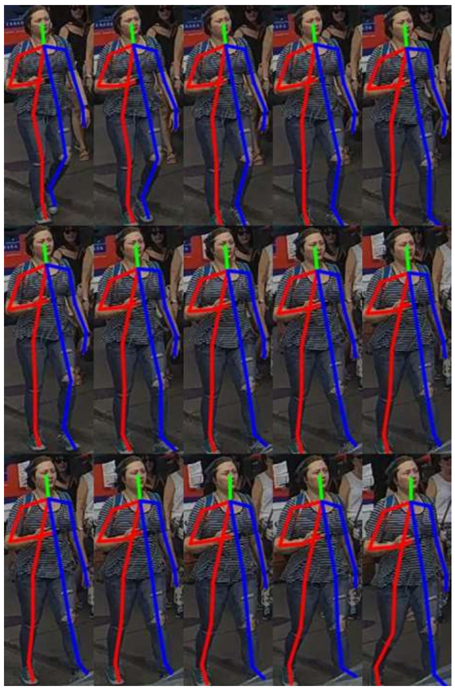
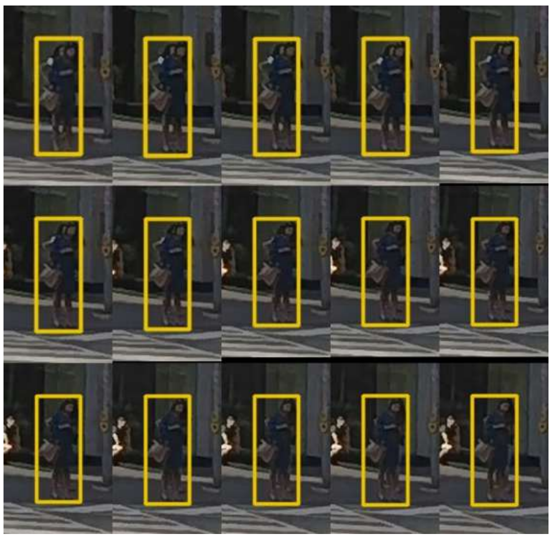
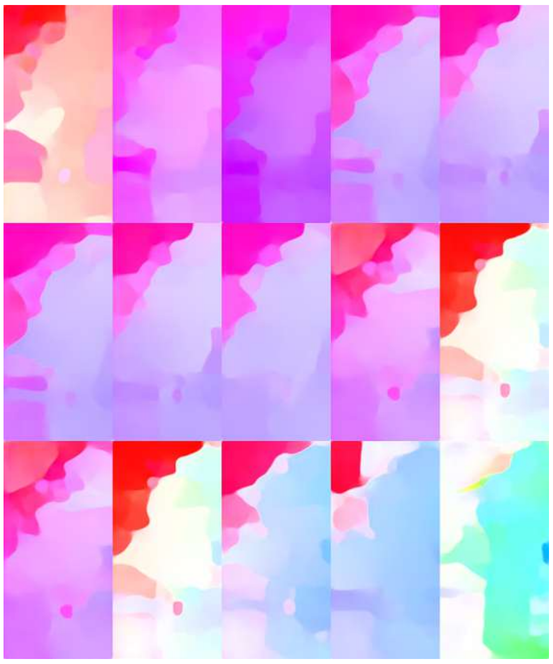

(d)

Figure A.7: Example non-crossing sequence 1.


(a)


(b)

(c)

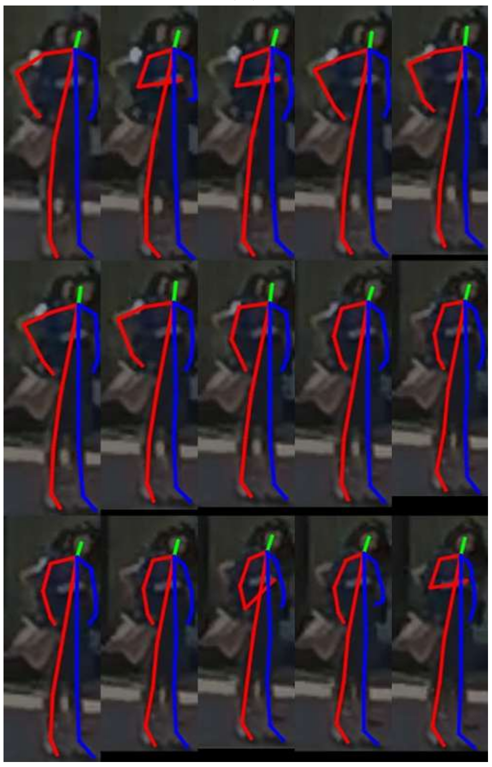
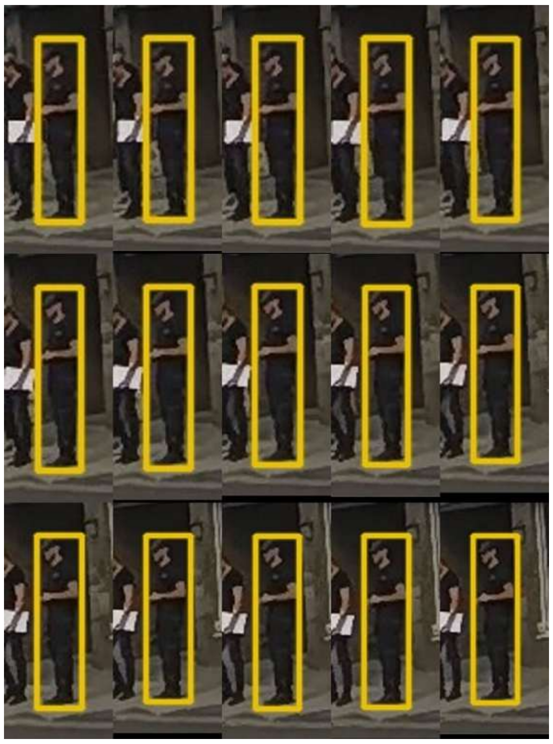
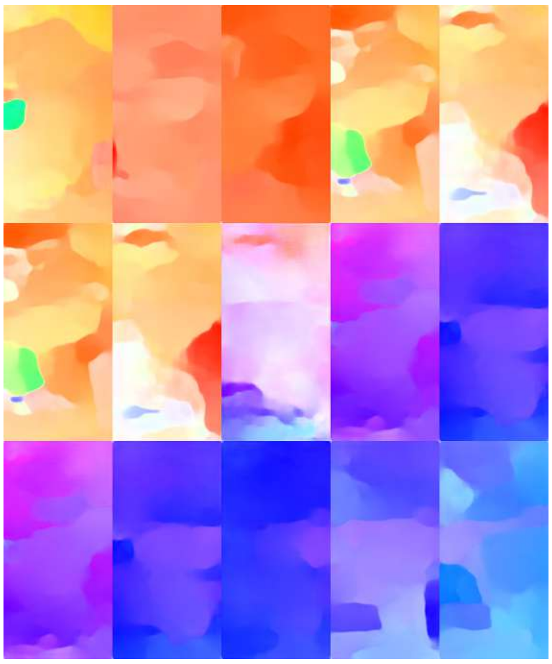

(d)

Figure A.8: Example non-crossing sequence 2.

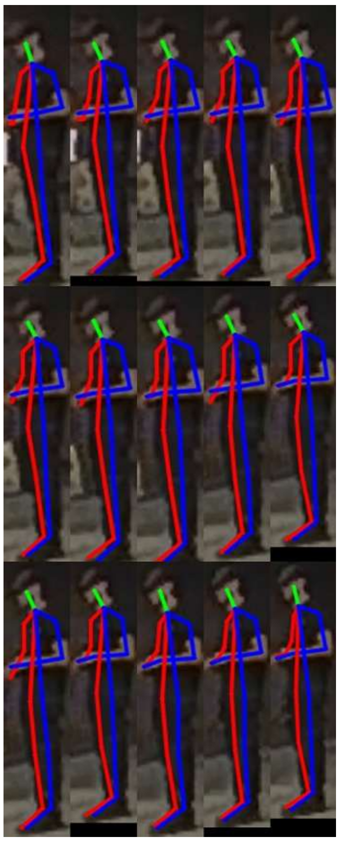


(a)



(b)

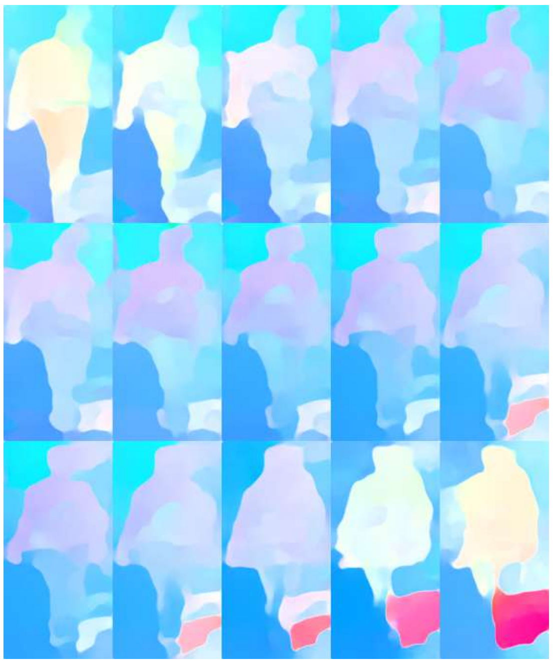
(c)

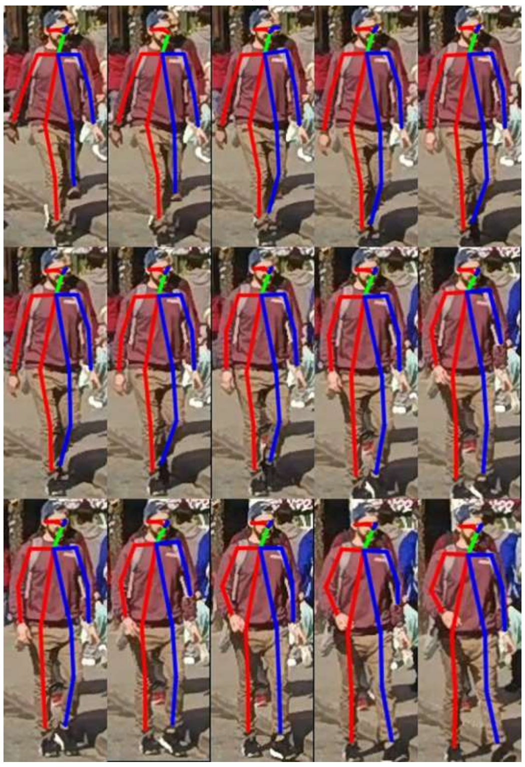


(d)

Figure A.9: Example non-crossing sequence 3.

(a)


(b)


(c)

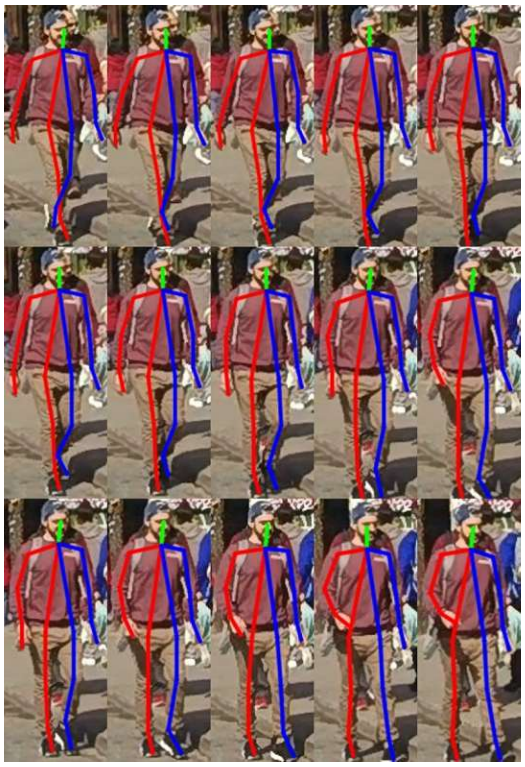

(d)

Figure A.10: Example non-crossing sequence 4.

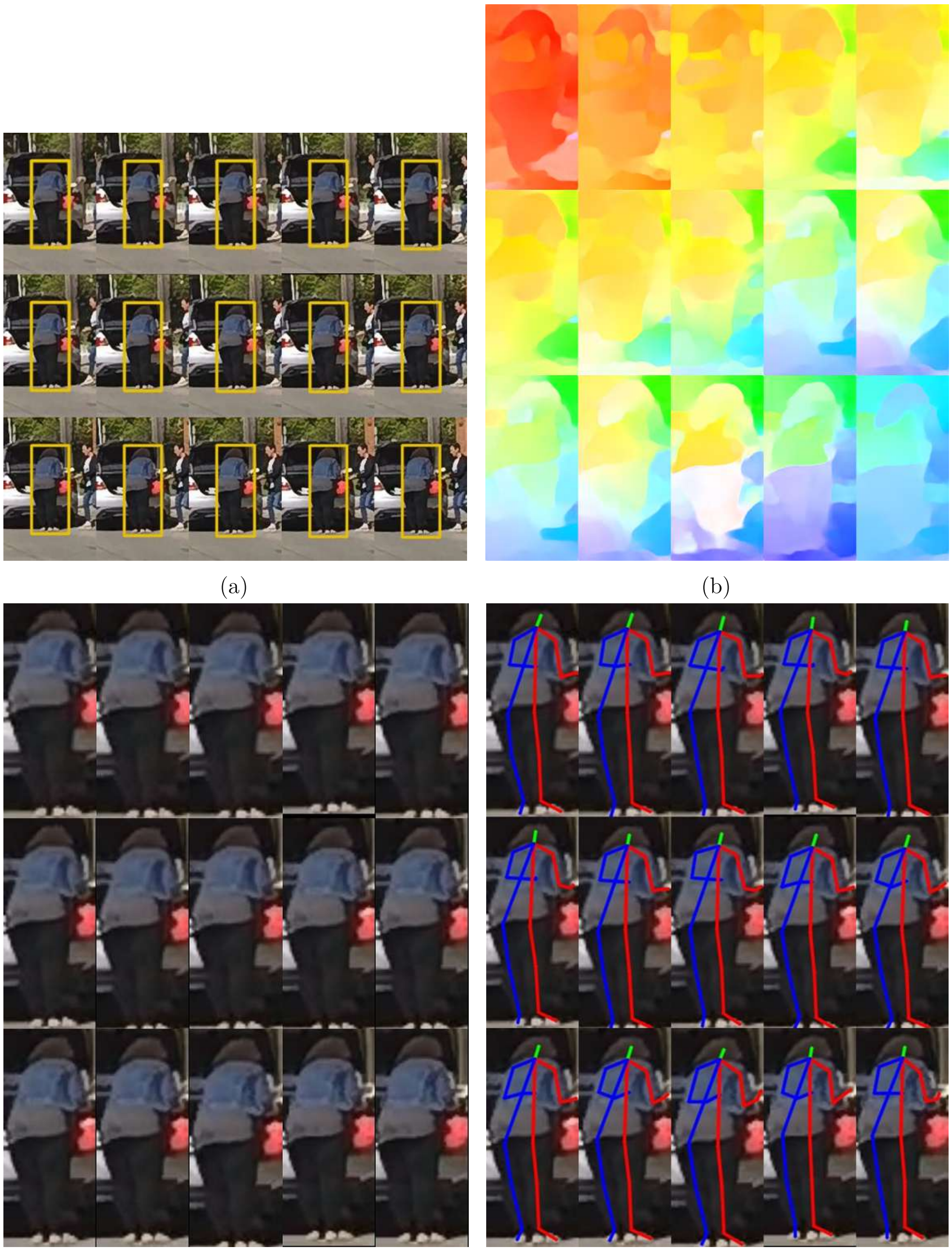
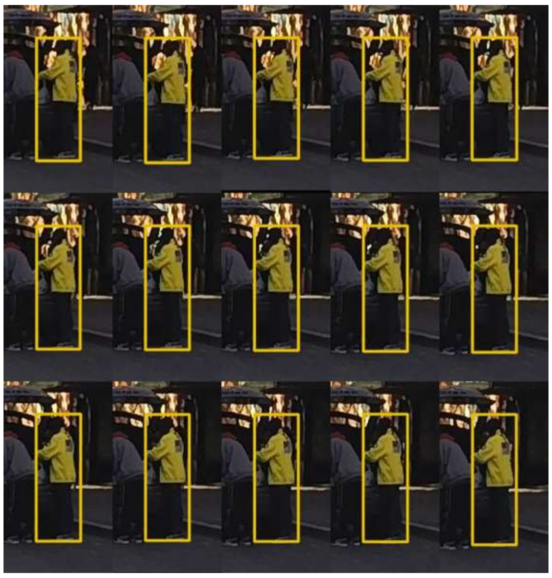
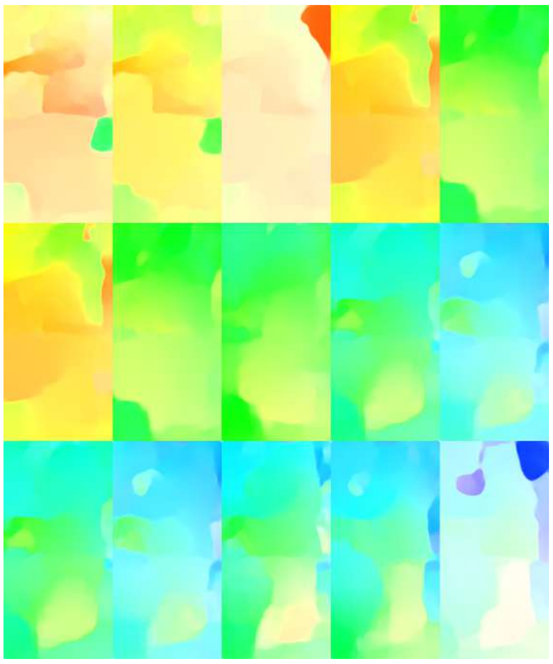
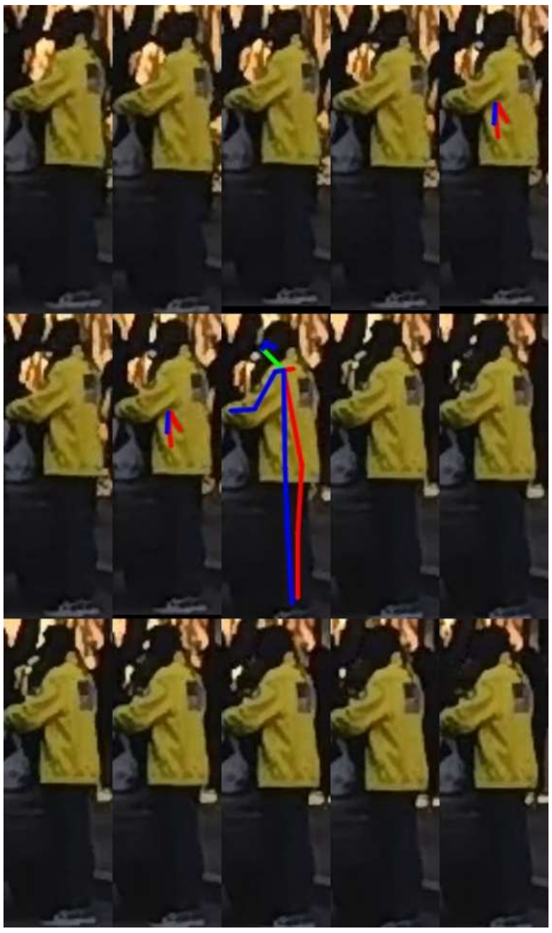

(a) (b)
(c) (d)

Figure A.11: Example non-crossing sequence 5.


(a)


(b)


(c)

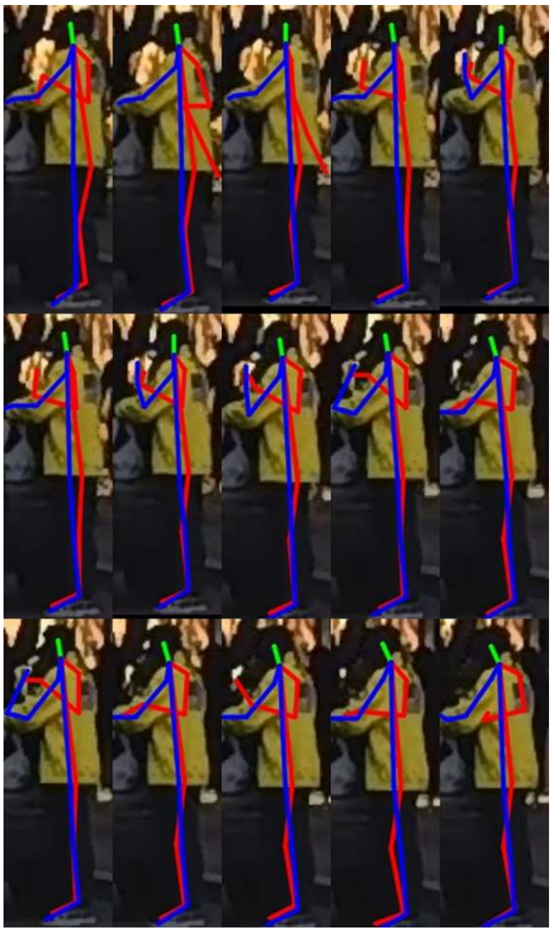

(d)

Figure A.12: Example non-crossing sequence 6.

List of Figures

List of Tables

Glossary

BDD100K is a dataset containing over 100k driving videos [YCW+20]. BDD is the abbreviation for Berkely Deep Drive, which is the consortium responsible for autonomous driving at the Berkely University. 25, 29, 40, 44, 49

COCO is the abbreviation for Common Objects in Context and is a popular dataset for different computer vision tasks like object segmentation/recognition and pose estimation [LMB+15]. 12, 15–17, 24, 26, 29, 40, 41, 43

Faster R-CNN is an object detector proposed by [RHGS16]. xi, xiii, 10–12, 20, 25, 26, 37, 39–41, 49, 69

FlowNet 2.0 is a neural network for estimating optical flow proposed by [IMS+17]. 9, 13, 14, 30, 33, 49, 53, 67

HRNet is the abbreviation for High Resolution Net, which is a top-down pose estimation network proposed by [XWW18]. xi, xiii, 17, 28–30, 44, 45, 49, 50, 53, 68

ImageNet is a large-scale dataset build for image classifaction [DDS+09]. 31, 32, 45

OpenPifPaf is a bottom-up pose estimation network proposed by [KBA21]. xi, xiii, 16, 17, 20, 26, 28, 30, 37, 39–46, 49, 69

OpenPose is a bottom-up pose estimation network proposed by [CSWS17]. 15–17, 26, 28, 30, 41–45, 53, 67, 68

ResNet is a network for image classification. ResNet stands for Residual Network and was proposed by [HZRS16]. 10, 11, 17, 32–34, 46, 68, 71

Simple Pose refers to the baseline pose estimation network provided in [XWW18], which uses a modified ResNet to detect the joints of a person. 17, 28, 30, 43–45

VGG is a neural network for image classification. It was proposed by the Visual Geometry Group [SZ15], hence the name. 16, 26, 31, 32, 45, 46, 49, 69

Acronyms

AP	Average Precision. 16, 17, 24, 43
AR	Average Recall. 24
CNN	Convolutional Neural Network. 8, 10–13, 15, 32, 33, 67
DTP	Dynamic Trajectory Predictor. xi, xiii, 9–11, 20, 32–34, 37–39, 49, 67–69
GRU	Gated Recurrent Unit. 9–11, 26, 27, 32, 33, 67, 68
JAAD	Joint Attention in Autonomous Driving. xi, xiii, 2, 3, 5, 6, 9, 20, 21, 27, 29, 32, 34, 37–39, 41–47, 49, 50, 67–69
MSE	Mean Squared Error. 35, 38
PAB	Pedestrian Action Benchmark. 20, 21, 26–31, 41, 42, 53
PIE	Pedestrian Intention Estimation. xi, xiii, 5, 6, 9, 21, 25, 27, 29, 31, 40–47, 49, 50, 53
RMSE	Root Mean Squared Error. 38, 69
RNN	Recurrent Neural Network. 9, 33
SF-GRU	Stacked with multilevel Fusion Gated Recurrent Units. 9, 26
SFRNN	Stacked with multilevel Fusion Recurrent Neural Network. xi, xiii, 20, 26–28, 31–33, 37, 38, 41–47, 49, 50, 68, 69
STED	Spatio-Temporal Encoder-Decoder. 10, 11, 31–33, 46, 67, 69

Bibliography

[AGR+16] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social LSTM: Human trajectory prediction in crowded spaces. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 961–971, 2016.

[AMF+21] Lina Achaji, Julien Moreau, Thibault Fouqueray, Francois Aioun, and Francois Charpillet. Is attention to bounding boxes all you need for pedestrian action prediction? *arXiv:2107.08031 [cs]*, October 2021.

[BWS05] Andrés Bruhn, Joachim Weickert, and Christoph Schnörr. Lucas/Kanade meets Horn/Schunck: Combining local and global optic flow methods. *International Journal of Computer Vision*, 61(3):1–21, 2005.

[CHMC21] Dooseop Choi, Seung-jun Han, Kyoungwook Min, and Jeongdan Choi. PathGAN: Local path planning with attentive generative adversarial networks. *arXiv:2007.03877 [cs]*, March 2021.

[CHS+21] Zhe Cao, Gines Hidalgo, Tomas Simon, Shih-En Wei, and Yaser Sheikh. OpenPose: Realtime multi-person 2D pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 43(1):172–186, 2021.

[CSWS17] Zhe Cao, Tomas Simon, Shih-En Wei, and Yaser Sheikh. Realtime multi-person 2D pose estimation using part affinity fields. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1302–1310, 2017.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009.

[Dea20] Jeffrey Dean. 1.1 The deep learning revolution and its implications for computer architecture and chip design. In *IEEE International Solid- State Circuits Conference (ISSCC)*, pages 8–14, 2020.

[DFI+15] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas

Brox. FlowNet: Learning optical flow with convolutional networks. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2758–2766, 2015.

[DOL20] Patrick Dendorfer, Aljoša Ošep, and Laura Leal-Taixé. Goal-GAN: Multimodal trajectory prediction based on goal position estimation. In *Asian Conference on Computer Vision (ACCV)*, pages 405–420, 2020.

[DTSB15] Alexey Dosovitskiy, Jost Tobias Springenberg, and Thomas Brox. Learning to generate chairs with convolutional neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1538–1546, 2015.

[Duc79] Claude E. Duchon. Lanczos filtering in one and two dimensions. *Journal of Applied Meteorology and Climatology*, 18(8):1016–1022, 1979.

[FL18] Zhijie Fang and Antonio M. López. Is the Pedestrian going to cross? Answering by 2D pose estimation. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1271–1276, 2018.

[GB14] Henry G. R. Gouk and Anthony M. Blake. Fast sliding window classification with convolutional neural networks. In *International Conference on Image and Vision Computing New Zealand (IVCNZ)*, pages 114–118, 2014.

[Gir15] Ross Girshick. Fast R-CNN. In *IEEE International Conference on Computer Vision (ICCV)*, pages 1440–1448, 2015.

[GM92] Melvyn A. Goodale and A. David Milner. Separate visual pathways for perception and action. *Trends in Neurosciences*, 15(1):20–25, 1992.

[GMB$^+$18] Omair Ghori, Radek Mackowiak, Miguel Bautista, Niklas Beuter, Lucas Drumond, Ferran Diego, and Björn Ommer. Learning to forecast pedestrian intention from pose dynamics. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1277–1284, 2018.

[GPB$^+$20] Joseph Gesnouin, Steve Pechberti, Guillaume Bresson, Bogdan Stanciulescu, and Fabien Moutarde. Predicting intentions of pedestrians from 2D skeletal pose sequences with a representation-focused multi-branch deep learning network. *Algorithms*, 13(12):331, 2020.

[HAGM15] Bharath Hariharan, Pablo Arbelaez, Ross Girshick, and Jitendra Malik. Hypercolumns for object segmentation and fine-grained localization. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 447–456, 2015.

[HS81] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1):185–203, 1981.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778, 2016.

[IMS$^+$17] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1647–1655, 2017.

[IS15] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning (ICML)*, pages 448–456, 2015.

[KBA19] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. PifPaf: Composite fields for human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11969–11978, 2019.

[KBA21] Sven Kreiss, Lorenzo Bertoni, and Alexandre Alahi. OpenPifPaf: Composite fields for semantic keypoint detection and spatio-temporal association. In *Transactions on Intelligent Transportation Systems (T-ITS)*, pages 1–14, 2021.

[KRT21] Iuliia Kotseruba, Amir Rasouli, and John K. Tsotsos. Benchmark for evaluating pedestrian action prediction. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1258–1268, 2021.

[KSFG14] Julian Francisco Pieter Kooij, Nicolas Schneider, Fabian Flohr, and Dariu M. Gavrila. Context-based pedestrian path prediction. In *European Conference on Computer Vision (ECCV)*, pages 618–633, 2014.

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. ImageNet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems (NIPS)*, volume 25, 2012.

[KSP20] Jeong-ah Kim, Ju-Yeong Sung, and Se-ho Park. Comparison of Faster-RCNN, YOLO, and SSD for real-rime vehicle type recognition. In *IEEE International Conference on Consumer Electronics - Asia (ICCE-Asia)*, pages 1–4, 2020.

[KTS$^+$14] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1725–1732, 2014.

[LAC$^+$20] Bingbin Liu, Ehsan Adeli, Zhangjie Cao, Kuan-Hui Lee, Abhijeet Shenoi, Adrien Gaidon, and Juan Carlos Niebles. Spatiotemporal relationship reasoning for pedestrian intent prediction. *IEEE Robotics and Automation Letters*, 5(2):3485–3492, 2020.

[LAE+16] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. SSD: Single shot multibox detector. In *European Conference on Computer Vision (ECCV)*, pages 21–37, 2016.

[LAI+21] Javier Lorenzo, Ignacio Parra Alonso, Rubén Izquierdo, Augusto Luis Ballardini, Álvaro Hernández Saz, David Fernández Llorca, and Miguel Ángel Sotelo. CAPformer: Pedestrian crossing action prediction using transformer. *Sensors*, 21(17):5694, 2021.

[LCCZ21] Wei Lu, Junyun Cui, Yanshuo Chang, and Longmei Zhang. A video prediction method based on optical flow estimation and pixel generation. *IEEE Access*, 9:100395–100406, 2021.

[LGG+17] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollar. Focal loss for dense object detection. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2980–2988, 2017.

[LH20] Renbao Lian and Liqin Huang. DeepWindow: Sliding window based on deep learning for road extraction from remote sensing images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:1905–1916, 2020.

[LK81] Bruce D Lucas and Takeo Kanade. An iterative image registration technique with an application to stereo vision. In *Imaging Understanding Workshop*, pages 121–130, 1981.

[LMB+15] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv:1405.0312 [cs]*, February 2015.

[LPWS20] Javier Lorenzo, Ignaco Parra, Florian Wirth, and Christoph Stiller. RNN-based pedestrian crossing prediction using activity and pose-related features. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 1801–1806, 2020.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3431–3440, 2015.

[LZD20] Shengchao Li, Lin Zhang, and Xiumin Diao. Deep-learning-based human intention prediction using RGB images and optical flow. *Journal of Intelligent & Robotic Systems*, 97(1):95–107, 2020.

[MBW+20] Sandeep Manandhar, Patrick Bouthemy, Erik Welf, Gaudenz Danuser, Philippe Roudot, and Charles Kervrann. 3D flow field estimation and assessment for live cell fluorescence microscopy. *Bioinformatics*, 36(5):1317–1325, 2020.

[MCPA20] Taylor Mordan, Matthieu Cord, Patrick Pérez, and Alexandre Alahi. Detecting 32 pedestrian attributes for autonomous vehicles. *arXiv:2012.02647 [cs, eess]*, December 2020.

[MDC20] Srikanth Malla, Behzad Dariush, and Chiho Choi. TITAN: Future forecast using action priors. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 11183–11193, 2020.

[PKSS21] Daniel Pototzky, Matthias Kirschner, Azhar Sultan, and Lars Schmidt-Thieme. Training object detectors if only large objects are labeled. In *British Machine Vision Conference (BMVC)*, 2021.

[PRC$^+$19] Dănuţ Ovidiu Pop, Alexandrina Rogozan, Clement Chatelain, Fawzi Nashashibi, and Abdelaziz Bensrhair. Multi-task deep learning for pedestrian detection, action recognition and time to cross prediction. *IEEE Access*, 7:149318–149327, 2019.

[RDGF16] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, real-time object detection. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 779–788, 2016.

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An incremental improvement. *arXiv:1804.02767 [cs]*, April 2018.

[RGB$^+$20] Adithya Ranga, Filippo Giruzzi, Jagdish Bhanushali, Emilie Wirbel, Patrick Pérez, Tuan-Hung Vu, and Xavier Perotton. VRUNet: Multi-task learning model for intent prediction of vulnerable road users. *Electronic Imaging*, 2020(16):109–1–109–10, 2020.

[RHGS16] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, 2016.

[RKKT19] Amir Rasouli, Iuliia Kotseruba, Toni Kunic, and John K Tsotsos. PIE: A large-scale dataset and models for pedestrian intention estimation and trajectory prediction. In *IEEE International Conference on Computer Vision (ICCV)*, pages 6262–6271, 2019.

[RKT17] Amir Rasouli, Iuliia Kotseruba, and John K Tsotsos. Are they going to cross? A benchmark dataset and baseline for pedestrian crosswalk behavior. In *IEEE International Conference on Computer Vision Workshops (ICCVW)*, pages 206–213, 2017.

[RKT20] Amir Rasouli, Iuliia Kotseruba, and John K. Tsotsos. Pedestrian action anticipation using contextual feature fusion in stacked RNNs. In *British Machine Vision Conference (BMVC)*, pages 1–13, 2020.

[RMA21] Haziq Razali, Taylor Mordan, and Alexandre Alahi. Pedestrian intention prediction: A convolutional bottom-up multi-task approach. *Transportation Research Part C: Emerging Technologies*, 130:103259, 2021.

[RRL$^+$18] Daniela Ridel, Eike Rehder, Martin Lauer, Christoph Stiller, and Denis Wolf. A literature review on the prediction of pedestrian behavior in urban scenarios. In *International Conference on Intelligent Transportation Systems (ITSC)*, pages 3105–3112, 2018.

[RRL21] Amir Rasouli, Mohsen Rohani, and Jun Luo. Bifold and semantic reasoning for pedestrian behavior prediction. In *IEEE International Conference on Computer Vision (ICCV)*, pages 15600–15610, 2021.

[SGS20] Olly Styles, Tanaya Guha, and Victor Sanchez. Multiple object forecasting: Predicting future object locations in diverse environments. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2020.

[SHN17] Khaled Saleh, Mohammed Hossny, and Saeid Nahavandi. Early intent prediction of vulnerable road users from visual attributes using multi-task learning network. In *IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, pages 3367–3372, 2017.

[SKS$^+$19] Amir Sadeghian, Vineet Kosaraju, Ali Sadeghian, Noriaki Hirose, Hamid Rezatofighi, and Silvio Savarese. SoPhie: An Attentive GAN for predicting paths compliant to social and physical constraints. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1349–1358, 2019.

[SRS19] Olly Styles, Arun Ross, and Victor Sanchez. Forecasting pedestrian trajectory with machine-annotated training data. In *IEEE Intelligent Vehicles Symposium (IV)*, pages 716–721, 2019.

[SXLW19] Ke Sun, Bin Xiao, Dong Liu, and Jingdong Wang. Deep high-resolution representation learning for human pose estimation. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5686–5696, 2019.

[SZ14] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, volume 27, pages 568–576, 2014.

[SZ15] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations (ICLR)*, 2015.

[TDT12] Cuong Tran, Anup Doshi, and Mohan Manubhai Trivedi. Modeling and prediction of driver behavior by foot gesture analysis. *Computer Vision and Image Understanding*, 116(3):435–445, 2012.

[WBP17] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. In *International Conference on Image Processing (ICIP)*, pages 3645–3649, 2017.

[WXW$^+$16] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision (ECCV)*, pages 20–36, 2016.

[XWW18] Bin Xiao, Haiping Wu, and Yichen Wei. Simple baselines for human pose estimation and tracking. In *European Conference on Computer Vision (ECCV)*, pages 472–487, 2018.

[YCW$^+$20] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving dataset for heterogeneous multitask learning. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2636–2645, 2020.

[YZY$^+$22] Dongfang Yang, Haolin Zhang, Ekim Yurtsever, Keith Redmill, and Ümit Özgüner. Predicting pedestrian crossing intention with feature fusion and spatio-temporal attention. In *IEEE Transactions on Intelligent Vehicles (T-IV)*, 2022.

[ZAZC20] Haitian Zeng, Haizhou Ai, Zijie Zhuang, and Long Chen. Multi-task learning via co-attentive sharing for pedestrian attribute recognition. In *IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6, 2020.

[ZF14] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In *European Conference on Computer Vision (ECCV)*, pages 818–833, 2014.

[ZSGY19] Zhengxia Zou, Zhenwei Shi, Yuhong Guo, and Jieping Ye. Object detection in 20 years: A survey. *arXiv:1905.05055 [cs]*, May 2019.

[ZTF11] Matthew D. Zeiler, Graham W. Taylor, and Rob Fergus. Adaptive deconvolutional networks for mid and high level feature learning. In *IEEE International Conference on Computer Vision (ICCV)*, pages 2018–2025, 2011.