CTU

CZECH TECHNICAL UNIVERSITY IN PRAGUE

F3

Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Auto-labelling of pedestrian road crossing from a monocular camera

Jonáš Koditek

May 2024
Supervisor: Ing. Lukáš Neumann, Ph.D.

BACHELOR‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Koditek Jonáš**	Personal ID number:	**507389**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Cybernetics**		
Study program:	**Cybernetics and Robotics**		

II. Bachelor’s thesis details

Bachelor’s thesis title in English:

Auto-labelling of pedestrian road crossing from a monocular camera

Bachelor’s thesis title in Czech:

Automatická detekce chodců vstupujících do vozovky

Guidelines:

1. Review and familiarize yourself with state-of-the-art generic object detectors and image segmentation methods.
2. Select a suitable object detector and image segmenation method, preferably with an existing implementation, such as Detectron2 [1,2], and make sure you can run it on custom images.
3. Propose a method to automatically label pedestrians as “crossing/not crossing the road” using the selected object detector and additional sources of information, such as camera calibration, scene segmentation.
4. Manually create a small testing dataset of images pedestrians crossing the road, by adding “crossing/not crossing the road” annotation to randomly selected pedestrian annotations from KITTI dataset [3].
5. Train a classifier which predicts whether the pedestrian will cross the road in near future, e.g. in the next 2 seconds
6. Evaluate the proposed method using the manually created testing dataset.

Bibliography / sources:

[1] HE, Kaiming et al., “Mask R-CNN”. Proceedings of the IEEE international conference on computer vision. 2017. p. 2961-2969
[2] Wu et al., “Detectron 2”, online: https://github.com/facebookresearch/detectron2
[3] G Andreas et al. "Vision meets robotics: The KITTI dataset." The International Journal of Robotics Research 32.11 (2013): 1231-1237

Name and workplace of bachelor’s thesis supervisor:

Ing. Lukáš Neumann, Ph.D.   Visual Recognition Group FEE

Name and workplace of second bachelor’s thesis supervisor or consultant:

Date of bachelor’s thesis assignment: **19.12.2023**   Deadline for bachelor thesis submission: **24.05.2024**

Assignment valid until: **21.09.2025**

Ing. Lukáš Neumann, Ph.D.	prof. Dr. Ing. Jan Kybic	prof. Mgr. Petr Páta, Ph.D.
Supervisor’s signature	Head of department’s signature	Dean’s signature

III. Assignment receipt

The student acknowledges that the bachelor’s thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor’s thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt   Student’s signature

Acknowledgement / Declaration

I would like to express my sincere appreciation to everyone, who supported me throughout the process of completing this thesis. It presented severe challenges that would be much more difficult to overcome alone.

Firstly, I would like to express my genuine gratitude towards my supervisor Ing. Lukáš Neumann, Ph.D. His insightful guidance, combined with so many years of experience in the field, and his constructive feedback helped to shape this thesis throughout the time. During the times of hardships and challenges, he helped me to navigate through obstacles by clarifying the problems, explaining their causes, and offering new perspectives. I am also grateful for his patience and understanding during periods when the progress seemed to be at a standstill.

I would also like to thank my family for their support throughout my work on this thesis. Their uplifting support always filled me with fresh energy and provided new inspiration.

Additionally, I am thankful for the access to the Research Center for Informatics (RCI) cluster with computational resources, which significantly enhanced my ability to perform the computational tasks required for this research.

Abstrakt / Abstract

Zvyšování bezpečnosti chodců poblíž silnice je často diskutované téma mezi výrobci automobilů a regulačními orgány. Když chodec vstoupí do silnice, řidiči obvykle trvá dlouhou dobu, než zareaguje a učiní potřebné kroky k tomu, aby nehodě předešel. To lze však zlepšit tím, že se automobil přepne do tzn. stavu ostražitosti, ve kterém je brzdový systém dopředu připraven na prudké brždění, což snižuje celkový čas mezi rozpoznáním chodce na silnici a zastavením vozidla. Tato práce předvádí algoritmus zaměřený na retrospektivní analýzu zaběrů z přední kamery auta a automatické nalézání momentů, kdy chodci vstupují do vozovky. Takto označená data mohou dále sloužit k učení klasifikátoru zaměřeného na predikci, zda chodec v danou chvíli vstoupí do vozovky nebo ne. Algoritmus byl schopný na BDD100k datasetu najít případy chodců stupujících do vozovky v 70.75% všech případů, kdy se tak stalo. Následně byl s těmito daty naučen klasifikátor, schopný predikovat až 1.3 vteřiny dopředu, zda daný chodec vstoupí do vozovky nebo ne. Na ručně anotovaném datasetu dosáhl klasifikátor 70.08% přesnosti.

Klíčová slova: Autonomní řízení, Automatické označování dat, Klasifikace, Počítačové vidění, Detekce objektů, Semantická segmentace, Sledování objektů

Increasing the safety of pedestrians near the road is a highly discussed topic amongst the car manufacturers and regulators. When a pedestrian steps into a road, the driver usually takes a long time to react and act accordingly to prevent an accident. However, this can be improved by putting the car into an alert state in which the breaking system is prepared for a rapid breaking, decreasing the overall time between recognizing the pedestrian in the road and actually stopping the car. This work proposes an algorithm focused on a retrospective analysis of the recordings from the car's front monocular camera and automatically labeling scenes in which pedestrians step into the road. The labeled data can further serve for training a classifier aimed at predicting whether each individual intends to enter the road in the near moment. Leveraging the BDD100k driving dataset, the developed algorithm was able to label cases of pedestrians crossing the road with a recall of 70.75%. Additionally, a custom classifier was developed aimed at predicting whether a given pedestrian will step into the road in the next 1.3 seconds or less. The labeled data were used for training the classifier, leading to an accuracy of 73.08% on the ground truth manually annotated dataset.

Keywords: Autonomous Driving, Automatic Data-Labeling, Classification, Computer Vision, Object Detection, Semantic Segmentation, Object Tracking

Contents

1 Introduction **1**
1.1 Description of the problem 1
1.2 Goals of the thesis 1

2 Preliminaries **2**
2.1 Datasets 2
2.1.1 KITTY-360 2
2.1.2 BDD-100K 2
2.2 Object Detection 3
2.2.1 Detectron2 3
2.2.2 Mask R-CNN R-50 FPN . . 3
2.3 Object Tracking 6
2.3.1 Multi-Object Tracking (MOT) 6
2.3.2 ByteTrack 6
2.4 Semantic Segmentation 7
2.4.1 Segformer 8
2.4.2 EfficientNet 9
2.5 Loss Function 10
2.6 Optimizers 10
2.6.1 Stochastic Gradient Descent 10
2.7 Regularization 11
2.7.1 Dropout 11
2.7.2 Weight Decay 12
2.8 Image Preprocessing 12
2.8.1 Histogram Equalization . . 12
2.8.2 Standardization 13
2.8.3 Normalization 13
2.8.4 Padding 14

3 Training Data Preparation **15**
3.1 Detection of Pedestrians 15
3.2 Tracking Pedestrians 16
3.2.1 Tracking Using a Set of Rules 16
3.2.2 Tracking Using ByteTrack . 17
3.3 Determining Road Area and Pedestrian Classification . 17
3.3.1 Detection of Pedestrians within a Road Using a Trapezodial Method . 17
3.3.2 Implementing Semantic Segmentation for Road Segmentation 18
3.3.3 Classifying Whether a Person Is Inside a Road or Not 18
3.4 Data Creation Pipeline 20
3.4.1 Algorithm for Filtering and Selecting Data 20
3.4.2 Positive Data Selection . . 20
3.4.3 Negative Data Selection . . 21
3.4.4 Algorithm add-ons 21
3.5 RCI Cluster 22
3.5.1 GPU utilization 23
3.5.2 CPU utilization 23
3.6 Experiments and Evaluation . . 23
3.6.1 Validation Dataset 24
3.6.2 Evaluation of the Method Using trapezoid area 24
3.6.3 Evaluation of the Method Using Semantic Segmentation 25

4 Classifier Development **26**
4.1 Architecture 26
4.1.1 Loss Function and Optimizer 27
4.1.2 Regularization 27
4.2 Training Process 27
4.2.1 Image Preprocessing 27
4.2.2 Hyperparameters Tuning . 28
4.3 Performance Evaluation 34

5 Conclusion **37**

References **38**

Tables / Figures

3.1 Evaluation of the trapezoidal method for generating positive data . . . 24
3.2 Evaluation of the semantic segmentation method for generating positive data . . . 25
4.1 Classifier performance . . . 36

2.1 Mask-R-CNN-ResNet-50 architecture . . . 4
2.2 Residual block . . . 5
2.3 ResNet . . . 5
2.4 ByteTrack . . . 7
2.5 Segformer architecture . . . 8
2.6 Stochastic gradient descent vs Gradient descent . . . 11
2.7 Dropout visualization . . . 11
2.8 Histogram Equalization . . . 13
2.9 Padding . . . 14
3.1 Mask-R-CNN-ResNet-50 output . . . 15
3.2 Object tracking . . . 16
3.3 Classification inside/outside of the road - trapezoidal method . . . 18
3.4 Classification inside/outside of the road - success . . . 19
3.5 Classification inside/outside of the road - not a success . . . 20
4.1 Train metrics of the Adam optimizer . . . 30
4.2 Train metrics of the SGD optimizer . . . 31
4.3 Train metrics of the SGD optimizer with a declining LR . . . 32
4.4 Dropout rate change . . . 33
4.5 Validation loss and validation accuracy of models with different architectures . . . 34
4.6 Train metrics of the final model . . . 35

Chapter 1
Introduction

1.1 Description of the problem

Detection of pedestrians in the road is one of the crucial problems in autonomous driving. When a random person steps into a road, the autopilot needs to react accordingly in the best interest of both the driver and the pedestrian.

Before the advent of autonomous driving, computer vision was not utilized in this field. Nowadays, with advancements in sensors and cameras, the field of autonomous driving is rapidly growing. However, cars do not need to be fully autonomous. Many car manufacturers have already implemented various systems that assist drivers, helping them drive more efficiently and safely.

One example is an assisting system for breaking. During emergencies where the driver needs to swiftly stop the car in order to prevent an accident, he is often limited by factors such as slow reaction time of himself and slow response of the breaking system. The slow response of the breaking system can be improved by transitioning the car into an "alert" state, in which the breaking system becomes prepared for rapid breaking.

This thesis focuses on contributing to the development of a system, capable of identifying scenarios, in which the car should be put into the "alert" state. Specifically, this thesis focuses on developing an auto-labeling algorithm, aimed to retrospectively identify cases in which a pedestrian entered the road. by developing such algorithm it is possible to automatically process current driving datasets and create a custom training dataset with cases of pedestrians entering a road. With this newly created dataset, it is possible to subsequently train a classifier aimed at predicting whether an individual will enter the road in the near moment.

1.2 Goals of the thesis

This work specifically aims to:

- Develop a robust method for the creation of training data suitable for training a classifier. This classifier aims to detect pedestrians intending to step into the road.
- Manually create a small testing dataset containing images of pedestrians labeled as "crossing/not crossing the road".
- Train the classifier by leveraging the created data.
- Evaluate the proposed method using manually created testing dataset.

Chapter 2
Preliminaries

2.1 Datasets

In order to perform the necessary steps in this thesis, there is a need for data to work with.

2.1.1 KITTY-360

The dataset named KITTY-360 was chosen in the initial stage of the research. This dataset is accessible as an open source. It was created by faculty members of the Karlsruhe Institute of Technology and Toyota Technological Institute at Chicago. The dataset was created using a recording platform (a car) equipped with multiple types of sensors including grayscale cameras, color cameras, or LIDAR. The output images from the color camera were captured at 10 frames per second and then cropped to 1382 x 512 pixels. After the rectification, the frames were slightly smaller. Despite using only the front monocular camera data, the dataset comprises several categories of benchmark data. Some examples are stereo flow, optical flow, scene flow, depth completion, 2D/3D object detection, multi-object tracking, pixel-level segmentation benchmarks, and many more. The dataset was captured in the Karlsruhe area located in Germany. Because of this, the dataset contains data representing an average European infrastructure. [1]

2.1.2 BDD-100K

BDD-100K is a large-scale diverse dataset containing driving video data. It contains over 100 000 videos which makes it one of the biggest driving datasets that are publicly available for academic and research purposes. The data were obtained in a crowd-sourcing manner uploaded by thousands of drivers. Mainly, the data were captured in metropolitan areas in the United States such as San Francisco, Bay Area, Los Angeles and New York. The videos are split into 70 000 training videos, 20 000 testing videos and 10 000 validation videos. Many previous datasets suffered from the lack of variety in the data. However, BDD100K provides diverse scenes captured in different weather conditions as well as different times of the day. This makes it a suitable source of training data, since the trained models can benefit from the variety of scenes, become more robust, and generalize well. Each video is 40 seconds long with 30 frames per second captured in a high resolution (720p). The high-quality imagery is beneficial for tasks such as object detection and tracking which will be utilized further in the thesis. The data were captured only by monocular cameras attached to the cars' front windshield.

The dataset also provides variety of benchmark annotations. These annotations provide necessary information for supervised learning algorithms to train and learn from the visual content of the video frames. Some of the tasks are object detection, semantic segmentation, instance segmentation and more. [2]

2.2 Object Detection

Object detection is a process of identifying and classifying objects belonging to a certain class in an image. Object detection approaches are typically separated into two categories: generative methods and discriminative methods.

The generative methods consist of a pose variability model, an appearance model, and a background model. The pose variability model describes the range of potential poses an object can take in an image. The appearance model describes, how an object typically looks in the image. That includes characteristics such as shape, color, texture, or other features. The model for the background helps differentiate between the the objects of interest and a background.

The discriminative method typically consists of building a classifier that can differentiate (discriminate) between images (or sub-images) that contain an object belonging to a certain class and those that do not. The parameters of a classifier are adjusted during the training process performed to minimize the error between the predictions and the ground truth data.

Convolutional neural networks are widely used for this kind of task. They fall into the discriminative category. These networks consist of layers processing the image pixels and extracting image features together with applying nonlinear transformations such as max-pooling, normalization, dropout, skip connections, or activation functions. For object detection tasks there are two main designs. One and two-stage methods. Both use a sliding-window approach. In both cases, there are two parallel sub-networks. The first method is used to estimate the object's position in the image and the second is to classify the object. In other words, the window predicts whether a certain area contains an object and refines its position. The two-stage method first identifies regions that might contain the specified classes and then performs detailed position refinement and classification of these objects. In the one-stage method the sliding window classifier tries to make a prediction directly when processing a selected area. [3]

2.2.1 Detectron2

This open-source deep-learning framework/library was developed by Facebook AI Research (FAIR) primarily focused on building, training, and deploying state-of-the-art object detection, instance segmentation, and related models. It is built on top of the PyTorch deep learning library [4], thus leveraging PyTorch's flexibility, ease of use, and GPU acceleration capabilities. Detectron2 offers a modular architecture that makes it possible to assemble different components, to create custom object detection and segmentation models. It includes pre-trained models that have been trained on large-scale datasets such as COCO (Common Objects in Context) [5], enabling users to perform transfer learning by finetuning these models on their specific datasets. Detectron2 provides implementations of various state-of-the-art models. This framework also emphasizes efficiency and scalability. Some examples of this include distributed training across multiple GPUs and optimizations for faster training and inference. [6]

2.2.2 Mask R-CNN R-50 FPN

The Mask R-CNN R-50 FPN model [7] [8] trained on the COCO dataset from Detectron2 was selected for the detection of pedestrians due to its robustness and accurate performance. Its architecture, built on the ResNet-50 backbone fused with the feature pyramid network (FPN), offers a balance between computational efficiency and accurate performance. The following description is focused mainly on the part of architecture responsible for object detection (bounding boxes), and not the semantic segmentation masks, since it is the part of the model output that is used in this project.

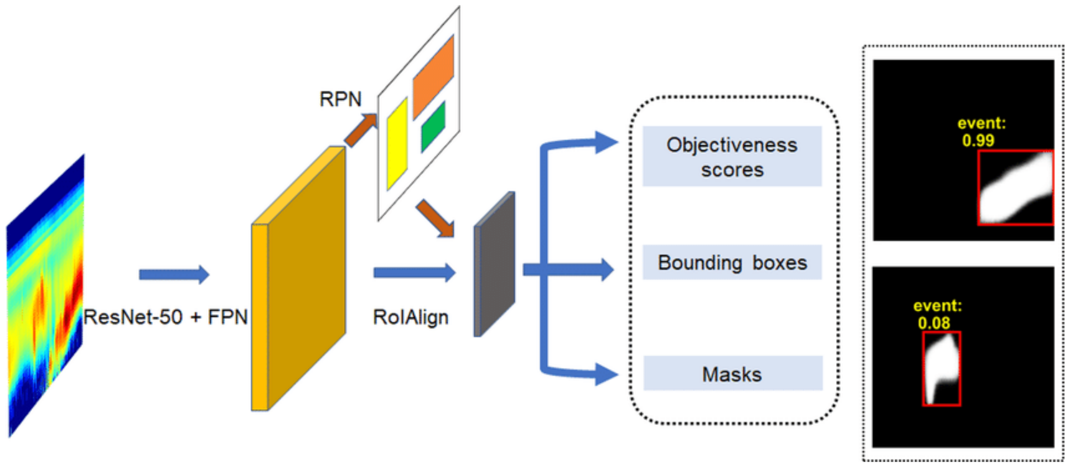


Figure 2.1. The architecture of Mask R-CNN ResNet-50 FPN. FPN is the feature pyramid network. RPN [9] is the region proposal network. RoIAlign (region of interest align) [10] is the layer that is properly aligning the features. (Image source: [11])

The Feature pyramid network (FPN) enhances the backbone by efficiently capturing features at multiple scales. By putting together information from different levels of the feature pyramid, the model provides effective detection of objects regardless of their size within the image. [12]

The region proposal network (RPN) [9] works as a two-stage method described in 2.2

The architecture also utilizes the ResNet-50 backbone. The number 50 in its name stands for the 50-layer depth. Together with residual connections, it is responsible for extracting hierarchical features from the input images. The residual connections are a fundamental component designed to address the issue of vanishing gradient during the training of very deep neural networks. It is represented as skip connections that introduce shortcuts to enable a smoother flow of gradients during training. Instead of attempting to learn the exact mapping of the input to the output at a certain layer, residual connections allow the network to learn residual functions, representing the difference between the input and the desired output.

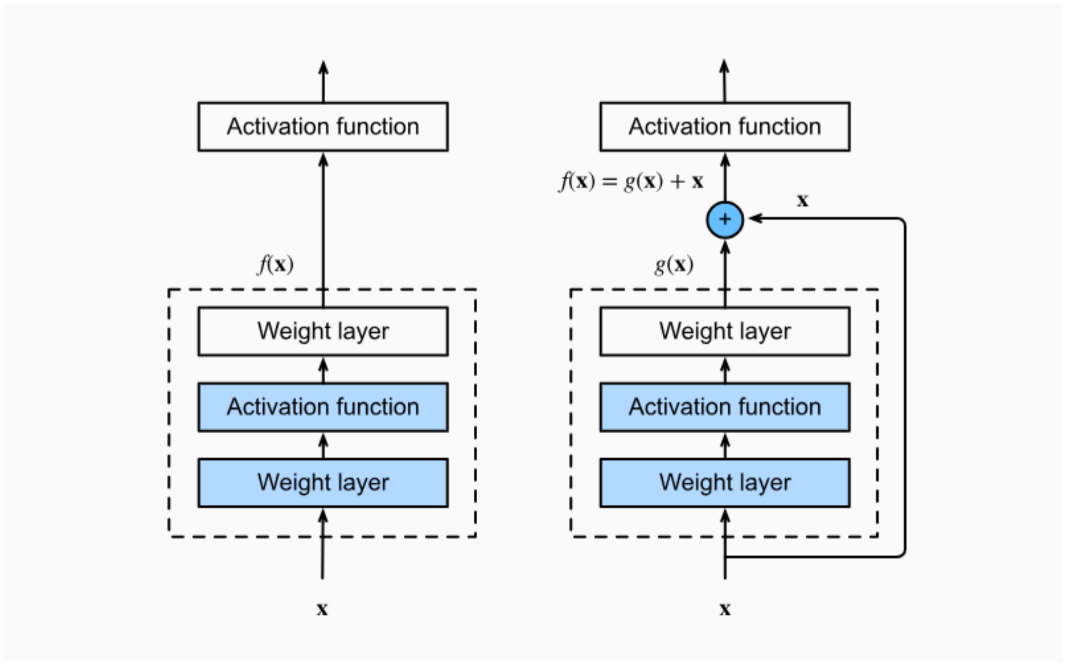


Figure 2.2. "*In a regular block (left), the portion within the dotted-line box must directly learn the mapping $f(\mathbf{x})$. In a residual block (right), the portion within the dotted-line box needs to learn the residual mapping $g(\mathbf{x}) = \mathrm{f}(\mathbf{x}) - \mathbf{x}$, making the identity mapping $f(\mathbf{x}) = \mathbf{x}$ easier to learn.*"[13] (Image source: [13])

In a typical neural network layer/block, the output is calculated as:

$$\text{Output} = \text{Activation}(\text{Weights} \times \text{Input}) \tag{1}$$

In a residual block (a building block of ResNet architectures), the output is formulated as:

$$\text{Output} = \text{Activation}_2(\text{Weights}_2 \times \text{Activation}_1(\text{Weights}_1 \times \text{Input}) + \text{Input}) \qquad (2)(2)$$

- **Input**: The original input to the block.
- **Weights$_1$**: Parameters of the first weight layer.
- **Weights$_2$**: Parameters of the second weight layer.
- **Activation$_1$**: First activation function (ReLU or another non-linear function).
- **Activation$_2$**: Second activation function (ReLU or another non-linear function).

Connecting multiple residual blocks in a series forms what is known as a Residual Network (ResNet) [14].

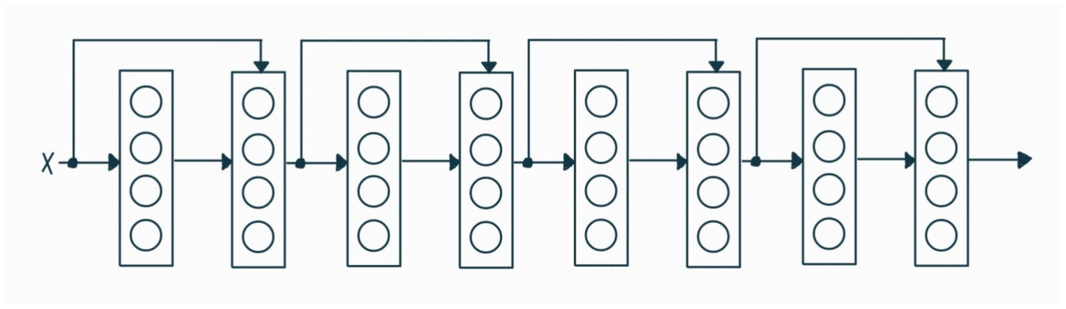


Figure 2.3. Simplified representation of ResNet. Each rectangle represents a layer consisting of the weight layer together with the activation function.

2.3 Object Tracking

Object tracking is a computer vision task aimed at locating and following objects across a sequence of images or a video stream. Each new object is assigned a unique ID and tracked across different frames of a video. This task has various different applications such as traffic monitoring, robotics, human activity recognition and also in autonomous vehicle tracking.

The task of tracking objects presents various challenges that are tackled by numerous algorithms. When designing such algorithm, there are certain aspects that are being considered. Some of them are listed below.

- Accuracy: Accuracy refers to how the tracking algorithm performs compared to ground truth benchmark data.
- Robustness: A robust algorithm is capable of tracking the target object even in bad conditions such as busy or cluttered background, occlusion, scale variation, similar appearance of objects and more.
- Real-time processing of information: Processing each frames takes certain time. The computational overhead presents challenges. When designing a tracking algorithm, there often arises a need for trade-off between accuracy and speed. [15]

There are multiple sub-tasks in the object tracking, each incorporating the tracking in a different way. Some of them are: single object tracking (SOT), multi-object tracking (MOT), visual object tracking and online multi-object tracking.

2.3.1 Multi-Object Tracking (MOT)

Multi-object tracking involves tracking more than one object in consecutive frames of a video. The object can be of the same or different class. The main challenge arising is that it is substantially harder to track multiple objects, especially in environments with multiple objects of the same class.

2.3.2 ByteTrack

ByteTrack is a powerful, yet simple state-of-the-art multi-object tracking algorithm. The algorithm combines outputs from an object detection algorithm together with a data association method named BYTE. Its main feature lies in keeping all the detected objects, including those with low detection scores. These are further used in the two-stage association process with the saved tracklets from the previous frame. tracklets[1] T are the output of the ByteTrack algorithm. They consist of the bounding box coordinates together with an ID of the object.

For each frame in a video sequence, new objects are detected using an object detection algorithm. The detected bounding boxes are separated into high confidence and low confidence ones based on the score threshold τ. Bounding boxes with a score below the low confidence are discarded. In the next step, a Kalman filter is used to predict new locations for tracklets T from previous frames.

During the first association stage, the high confidence bounding boxes detected in the current frame are associated with the tracklets T from previous frames (Including the lost tracklets, that are lost for less than a set number of frames). The matching is done using the linear assignment (Hungarian Algorithm [16]) that uses either IoU (intersection over union) or Re-ID feature distances. The second association stage works with the remaining unassociated tracklets from the previous frames and matches them

[1] Tracklets are only parts of the full tracks/trajectories of the examined objects

with the low confidence bounding boxes detected in the current frame. The matching is done in the same manner as in the first association stage, only the matching threshold is set lower. It is considered important to use the IoU in the second association stage when performing the linear assignment, since the low confidence bounding boxes often suffer from motion blur, occlusion and thus appearance features are not as reliable as in the first association stage.

To adress occlusion and disappearance of an object during the video, the remaining tracklets from the previous frames that were not matched with any of the bounding boxes in the current frame are considered as lost and stored as lost tracklets T for a defined amount of frames. If the object does not reappear and its corresponding tracklet happens to not match any new detection in this defined number of frames, it will be deleted.

The unmatched detection boxes from the current frame with a confidence lower than the score threshold τ are deleted. On the other hand, unmatched high confidence detection boxes are initialized as new tracklets [17].

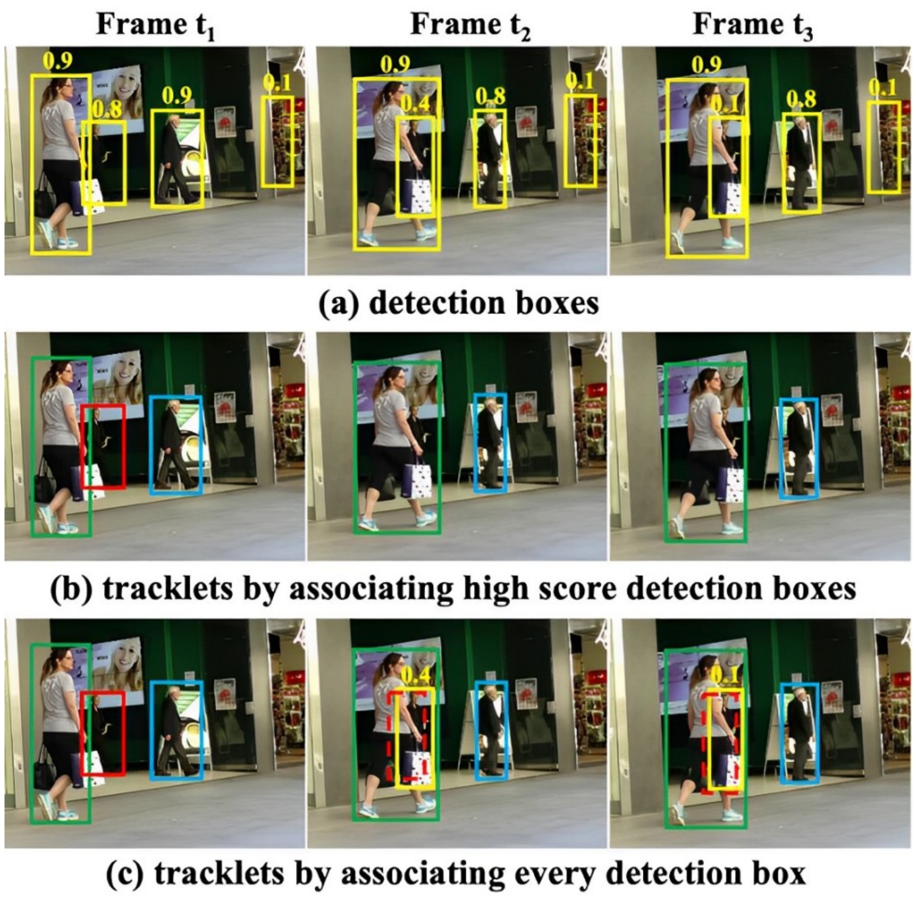

Frame t_1
Frame t_2
Frame t_3
0.9
0.8
0.9
0.1
0.9
0.4
0.8
0.1
0.9
0.1
0.8
0.1
(a) detection boxes
(b) tracklets by associating high score detection boxes
0.4
0.1
(c) tracklets by associating every detection box

Figure 2.4. ByteTrack uses both high and low score detections during reassociation stages. (Image source: [17])

2.4 Semantic Segmentation

Semantic segmentation is a subset of image segmentation. Its goal is to assign each pixel in an image a certain class. This makes it possible to differentiate between objects or regions within an image and provides a detailed understanding of the scene. A popular approach for this task is to use an encoder/decoder structure. The encoder down-samples the spatial resolution of the input image. This creates lower-resolution feature maps which proved to be very efficient for feature segmentation. On the other hand, the decoder up-samples the feature maps back into a full-resolution segmentation map [18].

2.4.1 Segformer

This framework from NVIDIA research combines a novel hierarchically structured Transformer encoder which outputs multiscale features, together with a simplified multilayer perceptron (MLP) decoder.

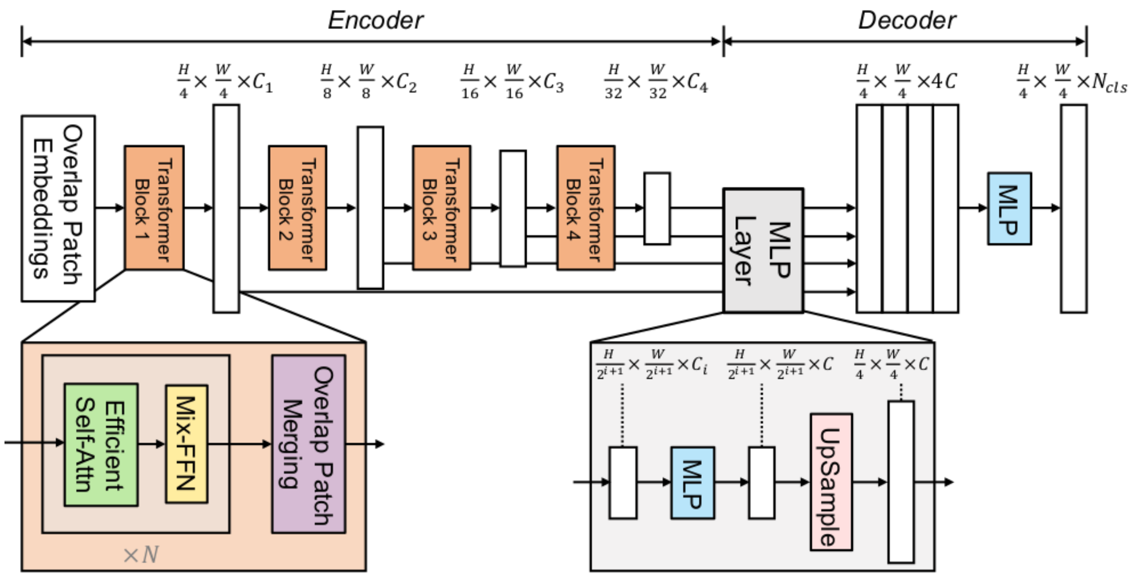

Encoder
Decoder
$\frac{H}{4}\times\frac{W}{4}\times C_1$
$\frac{H}{8}\times\frac{W}{8}\times C_2$
$\frac{H}{16}\times\frac{W}{16}\times C_3$
$\frac{H}{32}\times\frac{W}{32}\times C_4$
$\frac{H}{4}\times\frac{W}{4}\times 4C$
$\frac{H}{4}\times\frac{W}{4}\times N_{cls}$
Overlap Patch Embeddings
Transformer Block 1
Transformer Block 2
Transformer Block 3
Transformer Block 4
MLP Layer
MLP
Efficient Self-Attn
Mix-FFN
Overlap Patch Merging
$\times N$
$\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C_i$
$\frac{H}{2^{i+1}}\times\frac{W}{2^{i+1}}\times C$
$\frac{H}{4}\times\frac{W}{4}\times C$
MLP
UpSample

Figure 2.5. The proposed SegFormer framework consists of two main modules: A hierarchical Transformer encoder to extract coarse and fine features, and a lightweight All-MLP decoder to directly fuse these multi-level features and predict the semantic segmentation mask. “FFN” indicates a feed-forward network. (Image source: [19])

At its core, Segformer leverages transformer-based architecture with vision transformers (ViTs) [20]. It gains the ability to capture long-range dependencies by applying self-attention mechanisms across input patches.

One of the key distinguishing features of Segformer is its ability to handle high-resolution images effectively. Traditional CNN-based models [21] often require a deeper architecture to capture contextual information. However, deeper networks often suffer from the vanishing gradient problem. As a result, CNNs struggle to effectively propagate and update gradients across numerous layers, making it challenging to achieve good results in larger images. Segformer, on the other hand, divides an input image into smaller patches and applies attention mechanisms across these patches.

Moreover, this framework employs a hierarchical architecture that combines local and global features. The model extracts detailed local information from smaller patches while also capturing broader context from the entire image through a multi-scale processing. This significantly enhances the model's ability to comprehend how different parts of the picture are related to each other in a more detailed manner. Therefore the model is able to handle different sizes of input images compared to the ones it was trained on.

The simplified MLP Decoder is less complex. It gathers information from various layers of the transformer-based encoder, utilizing both local and global attention mechanisms. This approach results in strong representations[2] without needing an overly complex decoder [19].

2.4.2 EfficientNet

EfficientNet is a convolutional neural network architecture that has been designed to achieve state-of-the-art performance while maintaining high computational efficiency.

The core idea behind EfficientNet lies in its efficient scalability. In the original paper, the authors proposed a new scaling method called "compound scaling". this method uses a coefficient ϕ that uniformly scales the network's depth, width and resolution in the following manner:

$$\begin{aligned} \text{depth: } d &= \alpha^{\phi} \\ \text{width: } \omega &= \beta^{\phi} \\ \text{resolution: } r &= \gamma^{\phi} \\ \text{such that } \alpha \cdot \beta^2 \cdot \gamma^2 &\approx 2 \\ \alpha \geq 1, \beta \geq 1, \gamma &\geq 1 \end{aligned} \tag{3}$$

The α, β, and γ are constants that can be determined by a small grid search. The ϕ is a coefficient defined by user, used for defining the amount of resources available. Hence, the higher the ϕ, the more resources are available and thus allowing for an overall bigger architecture scaled in all depth, width and input image resolution. The α, β and γ parameters specify, how to distribute the compute resources across all three dimensions.

The idea of scaling all three dimensions at once comes from an intuitive idea that when an input image is bigger, the network needs more channels to capture more patterns that are present in the higher amount of pixels an image has. It also needs more layers to increase its receptive field.

Another key idea behind EfficientNet's architecture lies in using multi-objective neural architecture search (NAS) techniques, specifically AutoML [22], that automatically search and evaluate a large search space of candidate architectures to identify the most effective configurations.

Putting this together, the baseline EfficientNet model, namely EfficientNet-B0, was designed using the neural architecture search, using the AutoML MNAS framework. The goal was to optimize both accuracy and computational efficiency. Instead of targeting specific hardware latency, the efficiency was measured in FLOPS (floating point operations per second). This ensures the model performs well across different hardware configurations. The model utilizes a mobile inverted bottleneck convolutional block similar to MobileNet V2 [23] but with added squeeze-and-excitation optimization.

The next models were created by leveraging the base Efficient-B0 model and using the robust scaling method. In the first step, the ϕ was fixed to $\phi = 1$ assuming twice more resources available. A small grid search was performed that found values $\alpha = 1.2$, $\beta = 1.1$, $\gamma = 1.15$ for the base Efficient-B0 model, with the constraint $\alpha \cdot \beta^2 \cdot \gamma^2 \approx 2$. In the second step, The α, β, and γ were fixed as constants and the baseline network architecture was scaled by changing γ, using the equations in (3). That made it possible to develop the EfficientNet-B1 to B7 [24].

[2] representations are learned features or patterns that the model extracts from the input data

2.5 Loss Function

The machine learning model learns to map inputs to outputs. In order to get better in this mapping, it needs to adjust its parameters. This is done with a help of a loss function that takes the predicted model output y_pred and compares it with the ground truth y_true. The choice of loss function has to match the type of the problem the model aims to solve [25].

2.6 Optimizers

Optimizer utilizes the computed loss in order to alter the model parameters with a goal to minimize the loss function. There are two main types of optimizers. Adaptive optimizers and gradient descent optimizers. The gradient descent optimizers such as Stochastic gradient descent (SGD) and Mini-batch gradient descent update the parameters based on gradients computed from the training data. On the other hand, the adaptive optimizers such as Adagrad, Adadelta, RMSprop, and Adam, are able to adjust (adapt) the learning rate during the training process based on past gradients [26].

2.6.1 Stochastic Gradient Descent

Unlike traditional Gradient descent, the SGD requires only a random sample from the dataset to compute the gradient, instead of a whole dataset. This introduces randomness to the learning process and also makes it both more memory and computationally efficient. This makes SGD suitable to be used on large datasets. The SGD is also faster, because it updates the parameters after each sample, compared to the classic Gradient descent that updates the parameters after processing a whole set of data.

The SGD starts by initializing random model parameters ω. Next, it chooses a random data sample, compares it with the ground truth and calculates the loss. Next, it calculates the gradient of the loss function with respect to each parameter ω. Because the gradient represents the direction of the fastest increase in the loss function value, the goal is to go in the opposite direction to reach the local minimum of the loss function. The formula for computing the Stochastic gradient descent in each iteration is as follows:

$$\boldsymbol{\omega} := \boldsymbol{\omega} - \eta \cdot \nabla Q(\omega) \tag{4}$$

where Q is the objective function, often represented only by the loss function, that is being minimized. However, the objective function can be also represented by a loss function together with a penalty term such as L2 norm. The $\boldsymbol{\eta}$ represents the learning rate that determines the size of steps taken when adjusting the model parameters ω. The formula shows how the vector of parameters $\boldsymbol{\omega}$ is being updated by subtracting the gradient of a loss function that is scaled by a learning rate from the parameters ω [27].

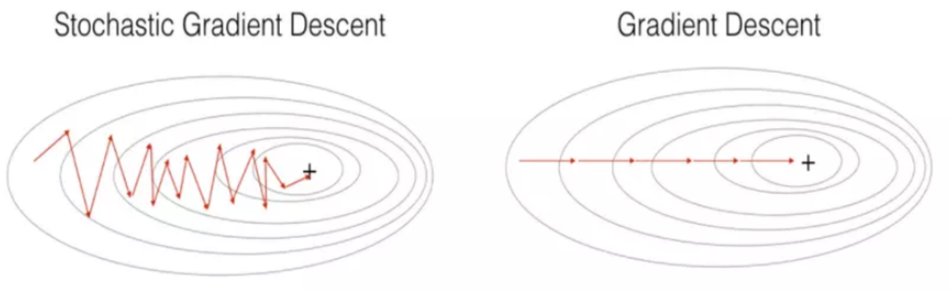


Figure 2.6. The Stochastic gradient descent iteratively updates the model parameters in order to reach the loss function minimum. It benefits from not having to process the whole dataset for computing the gradient. It converges quicker than a classic Gradient descent but is more prone to oscilations. This can be fixed by updating the learning rate. (Image source: [28])

2.7 Regularization

The regularization prevents the model from performing great on the training data but poorly on the testing and real-world data. It adds a penalty to the loss function to discourage the model from becoming overly complex, thereby promoting simpler models that generalize better.

2.7.1 Dropout

The dropout method temporarily drops out some of the nodes in the layer during the training process. This is done with a probability $1 -$ *drop probability*. The greater the drop probability, the more connections will be dropped. The overfitting happens, when the model learns the statistical noise. During training, the model parameters in each neuron/node are optimized to minimize the loss. However, some neurons can change in a way that they fix the mistakes of other neurons. This can lead to complex interactions between neurons and failure to generalize on unseen data. If the dropout is introduced, it prevents some neurons to fix the mistakes of other neurons. This is because in each batch/iteration, the connections between neurons are different and a presence of a specific neuron is highly unreliable [29].

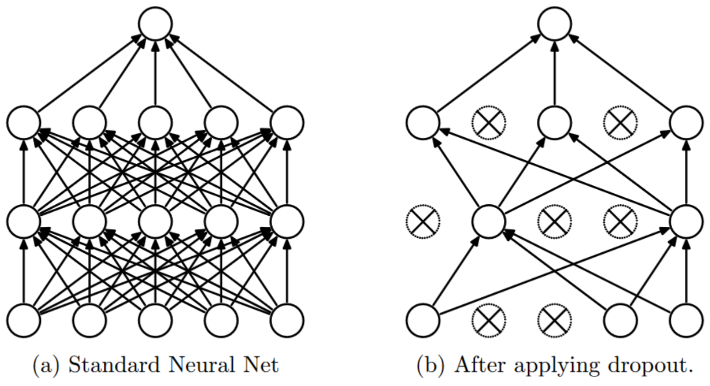


Figure 2.7. Dropout applied to a standard neural network. (Image source: [30])

2.7.2 Weight Decay

The main objective of the weight decay is to regularize the size of weights of parameters that are very large. It is also known as the L2 regularization, because it penalizes weights based on their L2 norm. By using the regularization technique, the objective function that is being minimized changes from just the loss function to a loss function together with an L2 norm of model's weights:

$$Q(\omega, b) = L(\omega, b) + \frac{\lambda}{2}\frac{1}{m}\sum_{i=1}^{m}\omega_i^2 \tag{5}$$

The $Q(\omega, b)$ is the objective function being minimized during the training process, together with weights ω and biases b as parameters. $L(\omega, b)$ is the loss function and λ is the weight decay parameter scaling the L2 norm of the weights.

The addition of the L2 norm forces the model to focus more on adjusting the smaller parameters and learning simpler functions that are less likely to lead to overfitting. During the training, the optimizer now uses a modified objective function such that the gradient is not only based on the training data, but also on the weight decay term [31].

2.8 Image Preprocessing

Image preprocessing is a crucial step in the computer vision tasks. It involves altering the raw image data into a format that is suitable for the model that is being fed with the data. The preprocessing steps have a significant influence on the model performance, reduce the influence of outliers and ensure the data are scaled the same.

2.8.1 Histogram Equalization

Historgram equalization is an image preprocessing technique used to improve the contrast in images. A color histogram of an image represents the number of pixels in each intensity level (a standard grayscale image has intensity levels ranging from 0 to 255). The histogram equalizer spreads the most frequent intensity levels. This method increases the global contrast of an image as the areas of lover contrast gain higher contrast [32].

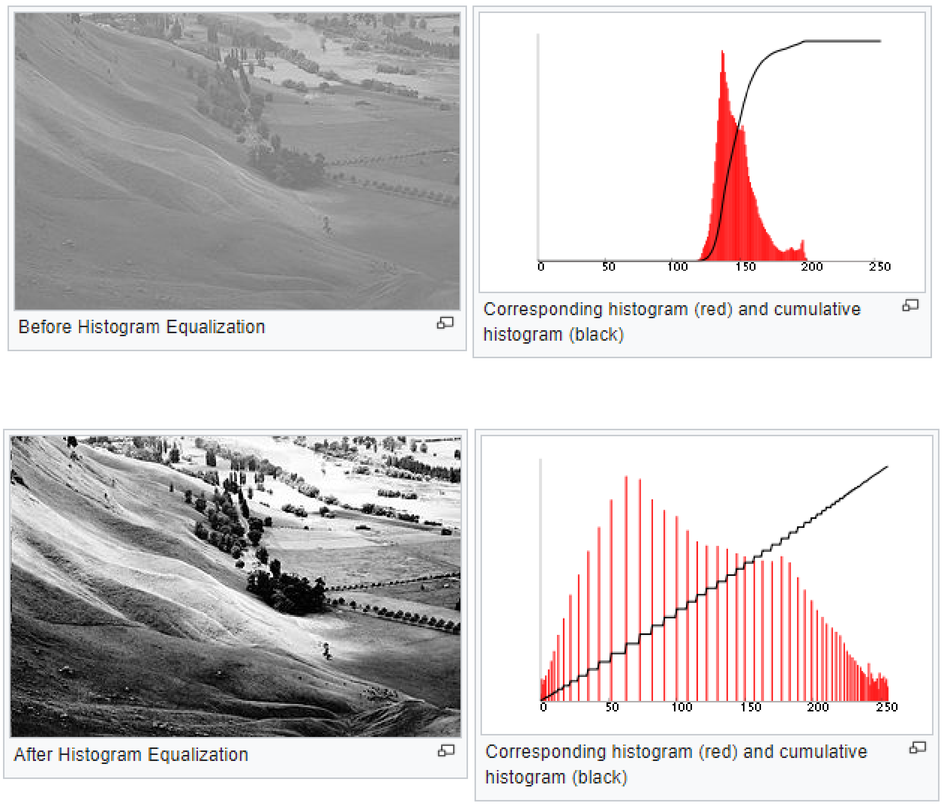


Figure 2.8. Histogram equalization applied to a grayscale image. (Image source: [32])

2.8.2 Standardization

Another preprocessing step, contributing to better model performance involves Standardization, also referred to as Z-normalization. It is very useful when the data features (pixel intensities) have large differences between their ranges. This process alters the data to have zero mean and the resulting distribution to have a unit standard deviation. Standardization is especially helpful, when the data follows Normal distribution, also known as the Gaussian distribution. The formula for standardization is:

$$X' = \frac{X - \mu}{\sigma} \tag{6}$$

In which X' is the altered image, X is the original image, μ is the meadian of image pixel values computed from the image or the whole dataset and σ is a standard deviation of pixel intensity values computed either from the single image or from the entire dataset. Computing the median and standard deviation values from the entire dataset provides better noise reduction and generalization by capturing the overall distribution of the data [33].

2.8.3 Normalization

Next technique is normalization. This method alters the data to bring the values of the features to a common scale. Specifically in this case, it involves scaling the features (pixel values) of the data to be inside a range between 0 and 1. This method is also known as min-max scaling. The formula for performing normalization is as follows:

$$X' = \frac{X - X_{min}}{X_{max} - X_{min}} \tag{7}$$

The X refers to an original image and X' to a modified image. The X_{min} and X_{max} represent the minimum and maximum ranges of values for a given image respectively [34].

2.8.4 Padding

Padding is a process of adding layers of zeros or other values outside the actual data in the input matrix. Its primary goal is to alter the image to the desired shape while preserving its content. A standard padding involves adding more layers to all left, right, bottom, and upper sides of the image symmetrically [35].

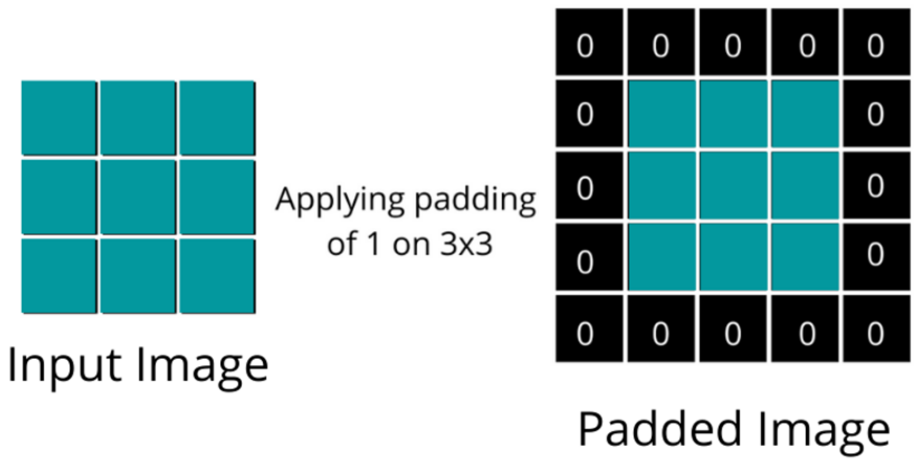


Figure 2.9. Example of applying a padding of size one on a 3x3 image. (Image source: [36])

Chapter 3
Training Data Preparation

3.1 Detection of Pedestrians

The first problem to solve was to detect pedestrians in each video frame. An optimal solution for this kind of problem was to use an object detection machine learning model. This type of model aims to identify and locate objects within images, thus automating the process. The selected model named `Mask R-CNN R-50 FPN` was firstly run on Google Colab, mainly because Google Colab provides a free GPU which is very handy. Because the dataset is just a sequence of images that form a video together, the detection has to be done on each frame separately. The Mask R-CNN R-50 FPN model takes a frame as an input and returns:

- Prediction classes - Each class is represented with an integer.
- Bounding box coordinates - They are of form (x1, y1, x2, y2) with (x1, y1) describing the upper left corner point and (x2, y2) being the lower right corner point of the bounding box.
- Prediction scores - This represents the probability of how likely the predicted object falls into the predicted class.
- Binary segmentation mask of the detected object.

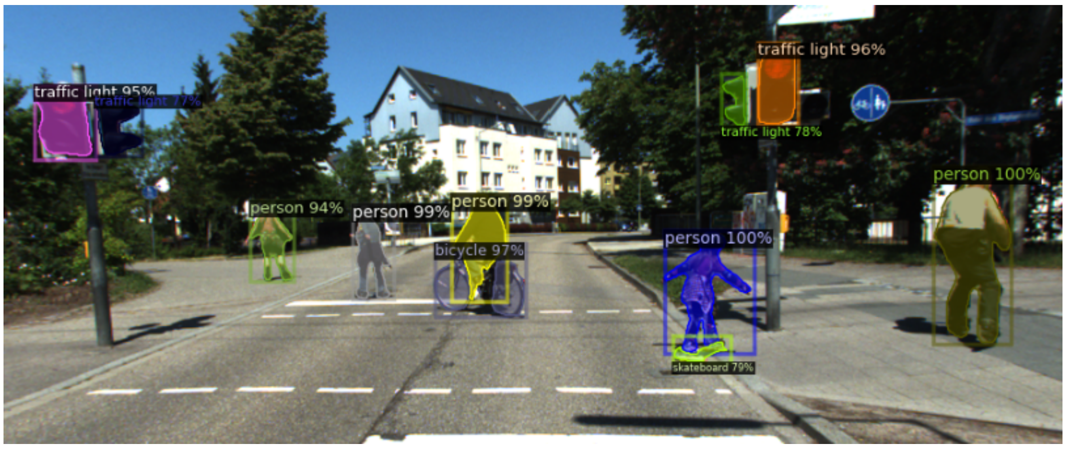


Figure 3.1. Output of the Mask R-CNN R-50 FPN model. Each object is assigned the most probable class together with being highlighted by a bounding box and a semantic segmentation mask.

The classes relevant to this task are the person, bicycle and motorcycle classes. The bicycle and motorcycle classes are needed in order to identify a cyclist and a motorcyclist in a frame so that they can be excluded from the analysis, since the focus is put only on pedestrians.

3.2 Tracking Pedestrians

Because the detection of individuals is performed on each frame separately, there is no direct way to know that the detected person on the next frame is the same as the one detected on the frame before. Therefore another problem to be solved emerged. Additionally the problem is more complex than tracking a single person, since all people in a video need to be tracked. This problem is called multi-object tracking (MOT) in the computer vision field.

3.2.1 Tracking Using a Set of Rules

This approach involves maintaining a dictionary that keeps track of detected pedestrians along with their unique IDs, which are continuously updated in each frame of the video sequence. Initially, every identified person is assigned a new ID. As each new frame is processed, a comparison is made between the newly detected people and all previously identified individuals from the preceding frame. By calculating the distance between the centers of bounding boxes of a newly detected person and a person from the previous frame, if the distance falls below a specific threshold, the algorithm considers the detected person to be the same individual as in the previous frame. The coordinates of the person with the corresponding ID in the dictionary are then updated. If any newly detected individuals were not associated with any of the existing IDs from the previous frame, they are considered as new figures, and new IDs are assigned to them.

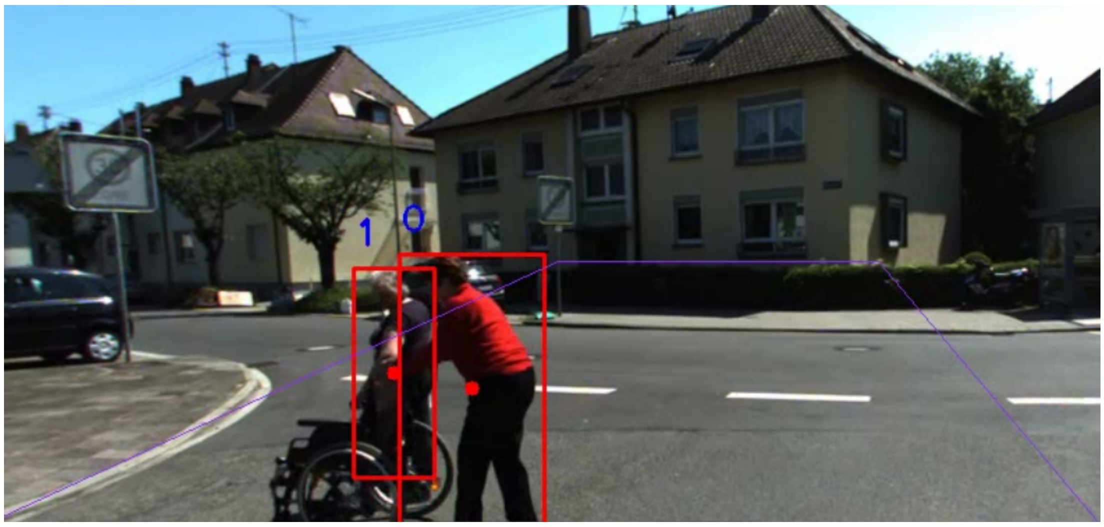


Figure 3.2. Pedestrians with corresponding IDs above their bounding box.

As it turned out, this method has two substantial downsides. The first occurs in scenarios when one individual occludes another. A typical scenario when this occurs is at a crosswalk when multiple people cross the road. Because this method only uses comparison based on distance, it is not possible to distinguish between two people where one occludes the other. Therefore the IDs of the two people can be assigned randomly and ID switches can occur. The second scenario occurs when a person gets out of the view of the camera and then reappears. This method automatically assigns a new, different ID when the person reappears again, which is not a desired behavior.

There are algorithms already developed that do not suffer from these inconveniences and that are able to solve these issues better. One of them is an algorithm called ByteTrack.

3.2.2 Tracking Using ByteTrack

As a second method, the state of the art multi-object tracking algorithm named ByteTrack was implemented. It addresses many of the shortcomings of the simplier rule-based tracking algorithm. Unlike the initial approach, ByteTrack utilizes more advanced techniques.

Compared to the initial tracking with a set of rules, the ByteTrack algorithm proved to be more robust. Even during low lightning conditions, the algorithm was able to correctly assign new detections to the existing tracklets. This is mainly due to the fact that the algorithm utilizes a two-stage association method. This enables reassociating even the low probability detections during the second association stage. When a person escapes the camera's field of view and then reappears again, the algorithm is able to assign the person the same ID again. This is done by keeping the lost tracklets for a certain period before they are discarded. During the experiments, ByteTrack excelled in scenarios where the tracked objects changed their position significantly across frames. This is largely due to the integration of Kalman filter that predicts the future position of objects based on their past motion.

During testing, the ByteTrack showed a limited performance in handling occlusions of pedestrians. Since the used implementation of the algorithm used only IoU during the linear assignment and did not leverage the appearance-based features, it was more prone to ID switches during these scenarios.

Despite the limitations, while the initial rule-based method mainly used the distance between bounding box centers for tracking, ByteTrack's use of both high and low probability detections, together with a sophisticated reassociation method resulted in significant improvement in tracking accuracy. The algorithm was better able to handle challenging scenarios and serve as a reliable component in the data creation pipeline described in section 3.4.

3.3 Determining Road Area and Pedestrian Classification

One of the core parts of this task is to accurately determine the location of the road area in frames. This part is crucial for determining whether the tracked individual is inside a road and represents a potential threat to the driver and themselves or if he is outside of the road where the risk of an accident with a car is not as high.

3.3.1 Detection of Pedestrians within a Road Using a Trapezodial Method

The first approach to detecting whether a pedestrian stepped into a road was by using a simple trapezodial method. The core idea is that a trapezodial area of a fixed size was defined for the separation of what is and what is not a road in an image. If the bottom middle point of the pedestrian bounding box was inside this trapezoid, the person was considered as being inside the road and vice versa. There is a function in the cv2 library [37] that is able to classify whether a given point lies inside a trapezoid. This function is called "cv2.pointPolygonTest" and it was applied in the function determining whether a given person stands inside the road.

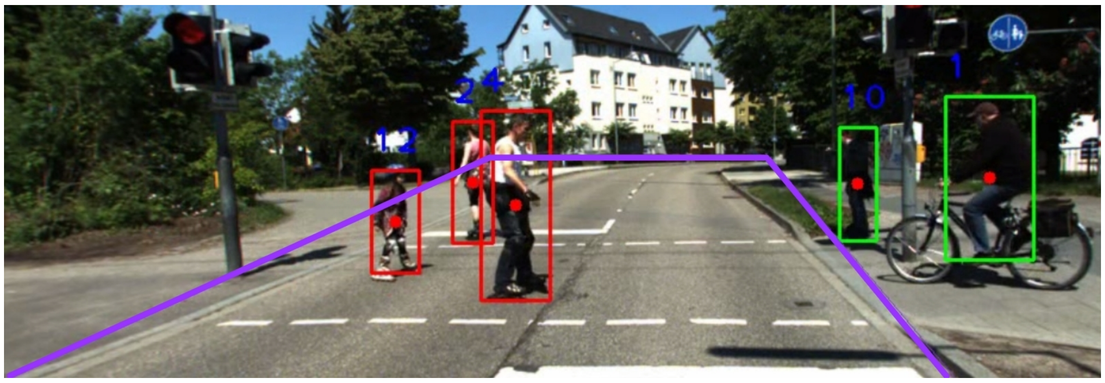


Figure 3.3. Classification of pedestrians inside (red) and outside (green) of the road. The trapezoid area is enclosed by the purple line.

This method has obvious drawbacks. Each road has a different width and structure. Therefore this solution is unable to generalize well. Another pitfall occurs during turns. The defined trapezoid does not match the road's curvature, leading to an incorrect classification of the road area. As it turned out, this method is prone to fail during other than usual scenarios. It fails to generalize across different road shapes and settings, thus making it a non-favourable option. This method initially served as a minimum viable solution that made it easier to work on other parts of the project.

3.3.2 Implementing Semantic Segmentation for Road Segmentation

Semantic segmentation stands out as a great tool for identifying the location of a road within an image. This provides the opportunity to combine the detection of pedestrians using an object detection model and segmented road using semantic segmentation. With these two results, it is possible to determine whether a pedestrian is inside/outside of the road. Because there are many semantic segmentation models already available, an established model was employed. Several semantic segmentation models were tested including Upernet from OpenMMLab. [38] The initial criterion was for the model to be compact enough for local execution. Unfortunately, it was not possible to reach the desired level of precision. Therefore larger model with better benchmarks had to be selected.

The next model employed was the Segformer developed by NVIDIA Research. Thanks to RCI cluster, it was possible to perform inference on the largest model Segformer-b5, that was trained on the Cityscapes dataset [39]. The PyTorch [4] model size is approximately 400 MB, comprising of the trained weights and biases of the model. The inference was performed on an NVIDIA Tesla A100 GPU utilizing RCI cluster, providing access to over 100 powerfull GPUs.

3.3.3 Classifying Whether a Person Is Inside a Road or Not

The road is represented by the class 0. The decision on whether a person is inside a road was based on whether the bottom middle point of the pedestrian bounding box intersects with the road segmentation mask. Unfortunately, during testing, there were many cases where this approach failed. The segmentation model also segments pedestrians, leading them to cover the road area.

As a result, the person segmentation mask can cover the road area and lead to a wrong classification. Thus, saying that the pedestrian is not inside the road, even though the person actually is inside the road. Only its segmentation mask covers the road area and the person's bounding box intersects with the person's segmentation mask. Hence, an alternative approach was required.

The new classification method whether a given pedestrian is within or outside of the road was based on two rules and at least one had to be met. Either 10% of the bounding box area had to contain pixels classified as a road or 20% of pixels in a small rectangle area around the bottom of the bounding box had to also be classified as a road. This method resulted in a sub-optimal performance during testing. In some cases, the classification inside/outside of the road failed because the segmentation output from the model was wrong compared to the ground truth. In other cases the approach failed because the pedestrian inside a road was surrounded by other objects (bike, scooter, wheelchair) that covered the road area during segmentation and therefore there were not enough pixels classified as a road around the pedestrian bounding box.

In the third method, a new solution was adopted. To tackle the issue of pedestrian segmentation mask covering the road segmentation mask, the algorithm analyzes a rectangle right under the person detection bounding box. The rectangle is of the same width and stretches 1/7 of the bounding box length down. If at least 50% of pixels in this area contain the road segmentation mask, the pedestrian is classified as being inside the road. This method significantly improved the performance and minimized the amount of false negative cases.

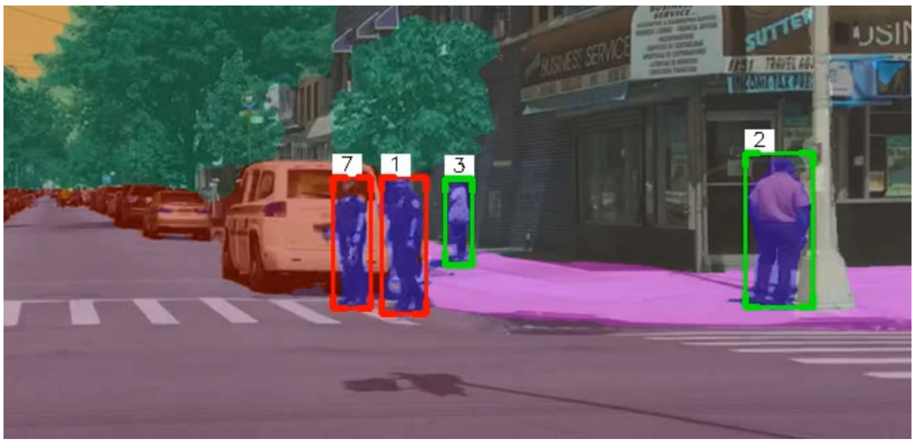


Figure 3.4. Classification of two pedestrians inside of the road. The road, sidewalk, and person classes are represented by purple, pink, and red colors respectively. In this figure, the pedestrians standing inside the road were successfully classified as inside the road.

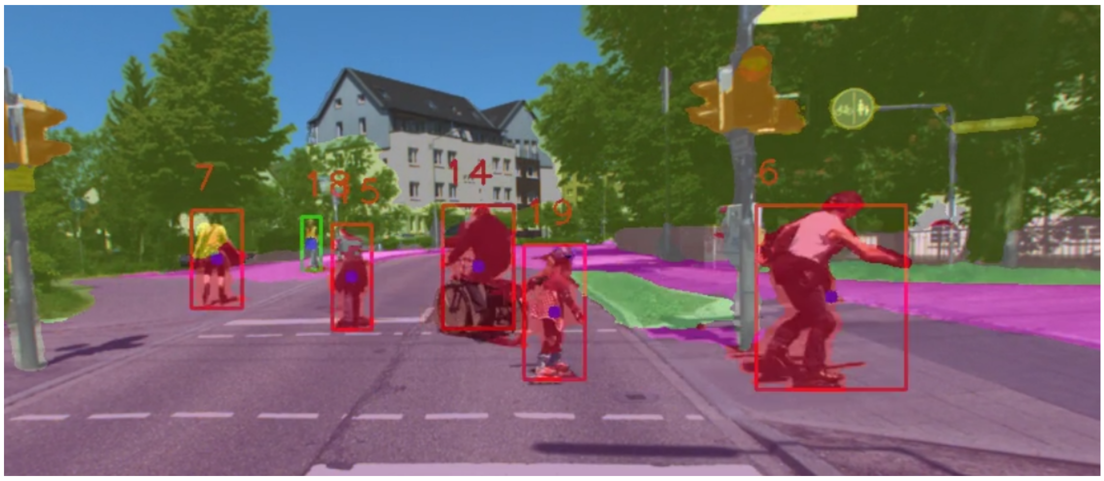


Figure 3.5. Classification of pedestrians inside (red) and outside (green) of the road. The road, sidewalk, and person classes are represented by purple, pink, and red colors respectively. In this figure both the right and left sides of the sidewalk are wrongly segmented as a road by the Segformer semantic segmentation model.

3.4 Data Creation Pipeline

The section 4 focuses on training a classifier, capable of predicting, how likely is a given person going to step into a road in the near moment. In order to be able to train such classifier, appropriate training data is needed. This section proposes a data creation pipeline that analyzes videos and outputs automatically labeled training data. The BDD100k dataset was chosen as a data source. Its contents comprising of 100 000 videos provide a wide scope of scenes, which make it easier to find appropriate training data. It contains scenes captured in different weather conditions as well as different times of the day. This ensures the extracted data will be diverse enough and make the classifier immune against failure thanks to its ability to generalize well.

The data at the output of the pipeline are in a form of a bounding box containing a person that is cut out of the original image. It is labelled as either a positive sample (the person is going to step into the road in the near future) or a negative sample (almost all other cases in which a person is not going to step into a road. This includes cases in which the pedestrian is already in the road, or where the pedestrian is going out of the road).

3.4.1 Algorithm for Filtering and Selecting Data

Since each video contains a different scene, it is necessary to analyze each video from the dataset separately. Initially, it is necessary to perform an object detection to know the exact position of every person in each frame. As a next step, a semantic segmentation is performed. This provides the necessary segmentation mask of a road, containing a pixel-wise information about which pixels in the frame are considered as a road. In order to classify, whether a person is going to step into a road, it is necessary to find the person in each frame of a video. For this, a tracking algorithm named ByteTrack was employed.

With the necessary information extracted from each frame, it is possible to classify whether a person is either directly inside or outside of the road. As explained in the section 3.3.3 the classification is performed by looking below the person's bounding box. If majority of the area below the person's bounding box is classified as a road, the person is considered to be inside the road and vice versa. This is performed for each person in each frame of a video to get the necessary information for further processing. This information is further utilized for determining, whether a given person is intending to step inside the road.

3.4.2 Positive Data Selection

The proposed method consists of counting the number of frames in which the person is inside or outside of the road. Firstly, the person has to be outside of the road for at least 40 frames. Because the classification of the road is not always accurate and the classification can be wrong, there can be up to three consecutive frames in which a person is classified as being inside the road. After exceeding this threshold, the count of the frames in which the person is outside of the road is reset to zero. This means the person has to be again outside of the road for at least 40 frames. On the other hand, if the person is inside the road in the next frame, the count targeting the number 40 continues. After reaching the desired threshold and the person is outside of the road for at least 40 frames, the algorithm checks, whether the person is inside the road for at least 30 frames. The same rule for interruption applies here. There can be up to three consecutive frames, in which the person is classified as being outside of the road. This accounts for mistakes in the road classification output in scenarios in which the person is very close to a road border. If these conditions are met, the person is then labeled as an individual that is going to step into the road.

Considering the task's objective of predicting, whether a person intends to step into a road, the most valuable data consist of frames right before a person's road entry moment. To determine the exact frame in which the pedestrian stepped into a road, one can simply get the frame in which the condition for being 30 frames inside the road was met and subtract the thirty frames together with all the frames in which an interruption occurred. To get the training data, one can simply extract frames, before the person steps into the road.

3.4.3 Negative Data Selection

In order for the classifier to be accurate, a same number of negative data samples is needed for training, thus a need for negative data arises. The negative data should represent the opposite. Those are simply moments in which a pedestrian is not planning to enter a road. Because of the nature of the task, the occurrence of pedestrians entering a road is usually low. This makes it much easier to obtain negative data. Considering the binary selection in which every non positive data frame would be a desired negative data frame, this approach presents a potential risk of extracting frames with moments very close to positive data frames or frames that have been just slightly missed during the extraction of the positive data. Therefore, an alternative approach was proposed.

Pedestrians detected outside of the road for more than 95% of their presence in the video were selected as valid negative data samples. The same condition was applied for pedestrians inside the road. Meaning if the person spent more than 95% of their presence inside the road, they were selected. In order to keep the dataset balanced, 40 random consecutive frames, featuring given pedestrians, were selected. The random selection of 40 frames was performed on a set of frames in which a given pedestrian manages to be tracked and appears in the video.

To introduce diversity into the negative data, an additional scenario was added. Individuals transitioning from inside the road outside, were deemed as suitable candidates for negative data. This condition involved verifying that a person remained inside the road for at least 45 frames, before moving outside of the road and staying there for at least 60 frames. The relaxing condition of allowing up to three frames of different classification, whether a person is or is not inside the road was not introduced in this case. Since there is way more cases with negative data, this condition was still met enough times.

3.4.4 Algorithm add-ons

As it turned out during the evaluation of created positive data, there was a number of cases in which the extracted bounding boxes did not meet the necessary quality requirements. Some bounding boxes contained just a part of the pedestrian, others were widely distorted from resizing to the uniform size. Some bounding boxes included cyclists and motorcyclists. To tackle these issues, new preprocessing methods were introduced.

Detections with too small bounding boxes showed low accuracy in representing person's features. Due to the smaller size of the bounding box, there was not enough pixels to represent the person accurately and thus making it challenging for further processing. To tackle this issue, a new threshold for a minimum bounding box dimensions was introduced. This conditions checks the width and height of the bounding box area with thresholds of 10 and 25 pixels respectively. This condition excludes bounding boxes that are too small.

In some cases, the bounding boxes did not contain the whole pedestrian's body. This happens in cases when the person is occluded by some object such as a car or other vehicle. This can also happen if the individual partially stands in a shadow. The object detection algorithm can wrongly annotate only the part of the body that is more visible and is not in the shadow. To address the problem, a condition for checking the aspect ration of the person's bounding box was introduced. A typical aspect ratio for a standing human being (width divided by the height) is between 0.35 to 0.5 regarding the results of the object detection algorithm. The checking condition filters out all the detected and tracked people, whose bounding box aspect ratio exceeds the value of 1 to eliminate the extreme cases.

Because the object detection algorithm detects only instances of discrete classes such as person, motorcycle and bicycle, there is no direct way for the algorithm, do distinguish between a pedestrian and a cyclist, motorcyclist or a person on any other vehicle. To tackle this issue, a new method filtering out cyclists and motorcyclists was proposed. The core idea of the method is to use IoU (intersection over union) of the person's and vehicle's bounding box. If the intersecting area is greater than a certain threshold, the person is deemed a cyclist or a motorcyclist and can be filtered out, based on the vehicle they are intersecting with. A standard approach would be to compare each bicycle/motorcycle bounding box with each person in the frame. To make the process less computationally demanding, a shortcut is introduced in a form of making the IoU calculation between the vehicle and the person that is closest to it. In order to find the closest person to a given vehicle, a KD-tree algorithm for the nearest neighbor was implemented.

The algorithm consists of constructing a KD-tree data structure which organizes points in multi-dimensional space into a binary tree. In this case, the points in the binary tree would be represented by the center-points of pedestrians' bounding boxes. Given a query point depicting the center point of the analyzed vehicle, a search in the KD-tree is performed [40].

After finding the nearest person and calculating the IoU with the vehicle, if the person is deemed a cyclist or a motorcyclist based on the intersection threshold, it is excluded from the list of potential individuals that are going too enter the road.

3.5 RCI Cluster

In order to carry out extensive computational tasks involved in data preparation and classifier training, an external source of compute power was needed. For this task, the RCI (Research Center for Informatics) cluster was chosen. It is a HPC (High Performance Computing) Infrastructure intended to foster the collaboration between fundamental scientists in application driven researchers from Faculty of Electrical Engineering, Faculty of Informatics and Faculty of Nuclear Sciences and Physical Engineering at CTU. The cluster is located at Charles Square at the Faculty of Electrical

Engineering. It consists of CPUs, GPUs and SMPs,[1] together with a data storage interconnected by a 100Gbit ethernet network. All the tasks are managed and scheduled by the SLURM [41] workload manager. This ensures that each GPU or CPU can be used by only one person at a time, preventing the unwanted interruption of workloads. The SLURM also ensures fair distribution of resources between users and job prioritization.

Regarding the specific hardware, there are original Intel nodes from 2019 and added AMD nodes from 2021. The Intel nodes comprise of Intel Xeon Scalable Gold CPUs and NVIDIA Tesla V100 GPUs. The AMD nodes consist of AMD EPYC CPUs and Tesla A100 GPUs.

3.5.1 GPU utilization

In order to speed up the object detection model, it was possible to utilize Cuda drivers that facilitate the communication between the CPU and the GPU. To enable the GPU acceleration, both model and data for processing have to be uploaded to the GPU. After this, it is possible to process the data using GPU, accelerating the processing speed by a significant factor.

The same process was applied to the Segformer semantic segmentation model. Since the model is built upon the PyTorch framework, it was necessary to upload both the model and the data to the GPU using Cuda drivers. To enhance the processing speed even further, batch processing was introduced. Instead of performing a model inference for each frame separately, a group of images was grouped together before being fed into the model. This method decreased the processing time by approximately 40%.

During the processing of the BDD100k dataset, the semantic segmentation procedure proved to be the most computationally demanding task. To tackle the still too long inference times considering the size of the dataset, the semantic segmentation was performed only on every second frame of each video. This meant that for a 30 fps video, the information about whether an individual resides inside or outside of the road was updated only 15 frames every second. The frames in between the segmented ones inherited the value from the previous frame.

To decrease the inference time even further, a smaller version of the Segformer model was used, namely Segformer-B3. This decreased the computational time significantly while preserving almost the same quality of the semantic segmentation output.

3.5.2 CPU utilization

The entire data creation pipeline was ran on the AMD EPYC 7543 CPU. It leveraged the already extracted information from the object detection and semantic segmentation models. Thanks to the workload manager and scheduler, it was possible to submit the processing task for the entire BDD100k dataset at once and the task scheduler automatically allocated CPU resources for the job.

3.6 Experiments and Evaluation

This section focuses on evaluating the developed data creation pipeline for auto-labeling pedestrian road-crossing. Considering the nature of the problem in which the goal is to find cases in which a pedestrian steps into a road, it is much more difficult to find the positive scenarios in which a person actually steps into a road. The conditions that have to be met are altogether quite complex and since there are not as much cases in

[1] symmetric multiprocessing units

which a person enters the road, the focus is centered on the algorithm for extracting the positive data. On the other hand, finding the negative cases in which a pedestrian does not enter the road are occurring several times more often. Because of that, it was possible to obtain enough false samples even with making the algorithm for extracting negative samples very strict. This ensured the algorithm extracted negative cases with a very high accuracy.

The following sections will focus on evaluating two distinct methods for data-labeling. Both utilize the Detectron 2 [6] for detecting pedestrians, ByteTrack [17] for tracking pedestrians and both utilize the same algorithms for positive and negative data selection. However, the first method utilizes the trapezoidal method for road area selection, while the second method leverages the semantic segmentation for detecting the road.

3.6.1 Validation Dataset

In order to quantify how accurately the data creation pipeline works, a small validation dataset was created, containing ground truth data extracted from the BDD100k dataset. It contains 106 positive cases in which the individual steps into the road and 106 negative cases in which the individual does not enter the road. Each positive case captures a moment of approximately 1.3 seconds or 40 frames before a person steps into a road. Each negative case also consists of 40 frames. This dataset will be used throughout this thesis in all experiments in order to ensure consistency of the results.

3.6.2 Evaluation of the Method Using trapezoid area

To evaluate the first method, all the 212 cases (106 positive and 106 negative cases) were taken from the validation dataset 3.6.1. These cases were then processed by the data creation pipeline using the auto-labeling algorithm for extracting cases in which the individual enters the road. The results were finally compared with the ground truth labels and relevant metrics were calculated. The method using the trapezoid for defining the road area demonstrated a poor performance. Since it classifies the road area solely on a predefined trapezoid, it cannot detect cases, such as pedestrian entering the road on the right side or the left side of a crossroad, in turns and many more. However, this method still managed to capture certain amount of the positive cases. Those were mostly cases where vehicles were driving along straight roads, particularly when individuals entered from designated crosswalks. The results are presented in the following table.

Type of Data	num. of cases [-]
True positives (TP)	24
False negatives (FN)	82
True negatives (TN)	98
False positives (FP)	8

Table 3.1. Evaluation of the trapezoidal method for generating positive data on the ground truth manually annotated dataset.

The obtained metrics are:

$$\text{Recall} = 22.64\%$$
$$\text{Precision} = 75.00\%$$
$$\text{F1 score} = 34.78\%$$

The results were obtained by running the first method for labeling positive cases in which pedestrians step into a road on the 106 positive ground truth cases and 106 negative ground truth cases. Out of the 106 positive cases, the method managed to capture 24 cases in which the pedestrian entered the road. This means the recall of the method is 22.64%. This reveals that this method struggles to find all the relevant cases effectively. Additionally, out of the 106 negative cases in which the individual does not step into the road, the algorithm selected 8 of those cases as positive. Hence, the resulting precision is 75%, highlighting that when the algorithm labeled a positive case, it was right in every three out of four cases. The F1 score combines the recall and precision and gives a more brief overview of how the algorithm generally performs.

3.6.3 Evaluation of the Method Using Semantic Segmentation

The second method tested involves utilizing semantic segmentation to extract the road area. This method provides a more detailed selection of the road area, making it possible to overcome the limitations of the initial trapezoidal method. To comprehensively evaluate the performance of this method, a similar evaluation process was used. Leveraging the small evaluation dataset, this method showed impressive results. Since it was able to leverage a more detailed information about the road, it captured cases that were previously missed by the trapezoidal method. The results are presented in the following table.

Type of Data	num. of cases [-]
True positives (TP)	75
False negatives (FN)	31
True negatives (TN)	103
False positives (FP)	3

Table 3.2. Evaluation of the semantic segmentation method for generating positive data on the ground truth manually annotated dataset.

The obtained metrics are:

$$\begin{aligned}\text{Recall} &= 70.75\%\\ \text{Precision} &= 96.15\%\\ \text{F1 score} &= 81.52\%\end{aligned}$$

The metrics obtained from the results signify an improvement in all areas. The precision increased to 96.15% compared to the previous 75% when using the trapezoidal method. This means that when the method extracts a positive case, it is right in 96.15% of the cases. The recall of 70.75%, compared to 22.43% for the trapezoidal method, demonstrates an increased performance in effectively capturing a large portion of all positive cases.

Chapter 4
Classifier Development

To leverage the developed auto-labeling algorithm and data creation pipeline and to put them into practice, a classifier was made. This section proposes a newly developed classifier focused on the binary classification of whether a given person is intending to enter a road or not. The classifier input comprises of a bounding box image containing a person. This image is ran through the classifier and the output is a probability distribution across two classes representing the likelihood of whether the person is intending or not intending to enter the road in the near moment. The model was developed using the PyTorch framework.[4] This framework was chosen because of its popularity amongst the research community. Its flexibility, ease of experimentation, and hardware acceleration support make it a great choice.

4.1 Architecture

In order to achieve better performance, it is possible to perform transfer learning. That involves leveraging an already trained model, that has been trained on a similar task. These models have been already trained on large datasets, which enables them to learn the underlying features well. In deep learning computer vision tasks, a model tries to learn the basic features in the early layers such as edges and corners. In the middle layers, it focuses on more complex patterns and shapes. In the final layers, the model learns features that are more task specific and contribute to the classification decision.[42] Because of this, it is possible to leverage the already trained features in the lower and middle layers and add a custom classifier head. This new model with a custom classifier head can be then fine-tuned on the task specific labeled data.

For this task, the EfficientNet model was selected as a base feature extractor. The model is pretrained on large-scale datasets such as ImageNet [43]. The model has already learned complex feature representations that can be generalized for the specific task of this thesis.

In order to tailor the desired model to the task requirements, a custom classifier head was designed. It is a substitute for the last fully connected layers in the EfficientNet model, which were removed. The image is initially processed by the modified EfficientNet model. Its output then goes into an adaptive average pooling layer. This layer serves for resizing the feature maps to smaller dimensions while trying to preserve the important features. This step is usually done at the end of convolutional neural networks to make it easier for the fully connected layers to process the data and reduce the computational complexity. The average pooling uses a kernel of certain size that slides over the image and computes the average of pixel values in the kernel window. This is done for each channel of the input image and reduces the spatial dimension of each channel.[44] Next, the output from the adaptive average pooling layer goes into a flattening layer. This layer takes the input multi-dimensional tensor and rearranges its elements into a one dimensional (1D) tensor so that it can be fed into subsequent fully connected layers. This is due to the fact that convolutional and pooling layers operate with multidimensional tensors, but the fully connected layers operate with one dimensional tensors. The flatten layer reduces the dimensionality of the feature maps. This leads to reducing the number of parameters in the subsequent fully connected layers which can reduce model complexity and make the model less prone to overfitting.[45] The flattened output then goes into the fully connected layer. Since using PyTorch framework, the fully connected layer is named as linear layer, referring to the linear transformations this layer performs.

The classifier head is made out of fully connected layers, preceded by a dropout that is applied during the training process and followed by an activation function. Each fully connected layer downsamples the dimensionality of the feature map. At the end of the network, the dimensionality is reduced to just two nodes, where each node represents one out of two classes: "person is going to enter the road soon" and "person is not going to enter the road soon".

4.1.1 Loss Function and Optimizer

The choice of loss function has to match the type of the problem the model aims to solve. For this classification task, the cross entropy loss proved to be the most suitable. It calculates the difference between the predicted probabilities and the ground truth labels for each class. To calculate the cross entropy loss, the formula is:

$$CE = -\frac{1}{N}\sum_{i} y_i - \log(y_{\text{pred}}) \tag{8}$$

in which y_i is the ground truth label and y_{pred} is the predicted label.

Regarding the optimizer, both Adam and SGD were tested during the implementation. The Adam optimizer did not perform poorly. However, the SGD performed better by a few percent as demonstrated in section 4.2.2. Because of that, the Stochastic gradient descent optimizer was selected. Additionally, the optimizer was enhanced by incorporating a learning rate scheduler that decreases the learning rate each epoch, making it easier for the model to converge to the local minimum by finding the parameters that minimize the loss function the most.

4.1.2 Regularization

In order to prevent overfitting, regularization techniques were employed. These prevent the model from performing great on the training data but poorly on the testing and real-world data. The first regularization technique was to use the dropout method described in section 2.7.1. Next, the weight decay described in section 2.7.2 was used.

4.2 Training Process

During the training process, the training data were extracted from the BDD100k dataset using the data creation pipeline presented in section 3.4. The extracted data were then split into 90% for training and 10% for validation.

4.2.1 Image Preprocessing

The preprocessing steps have a significant influence on the model performance, they reduce the influence of outliers and ensure the data are scaled the same.

Each image is initially augmented. This is done using several operations. One is horizontally flipping each image. Because the input image is a bounding box of a pedestrian, it does not matter whether the person is aiming to the right or to the left. This can reduce the chances of learning just the specific scenarios in which people always hold a certain position aiming to the same side. Next augmentation is done by cropping the pedestrian image and selecting just a part of the original image. This helps the model to better generalize by exposing it to a wider variety of appearances of a pedestrian within a bounding box. It can help the model to classify images, in which pedestrians may not be centered or fully visible. Another augmentation is introduced by adding a “jitter” to the images. This means changing the brightness, contrast, saturation and hue of the image. By incorporating these changes into the data, the model can become more invariant to these changes and thus be more robust. It is almost certain that the model will encounter scenarios with completely different lightning, weather, and environmental conditions.

Next, a histogram equalization is applied. The original image is in RGB format. However, histogram equalization can not be applied to each component as it leads to dramatic changes. To resolve this issue, the image can be initially converted to a HSV (hue, saturation, value) format and histogram equalization can be applied only to the value component, keeping the hue and saturation components untouched. The altered image can be subsequently converted back to the RGB format.[32]

Another preprocessing step, contributing to better model performance involves Standardization. The newly created training dataset extracted from the BDD100k dataset is very large and computing the mean and standard deviation for the whole dataset would be computationally demanding and difficult. Hence, the mean and the standard deviation for the ImageNet [43] dataset was used. The dataset comprises of images with similar features, therefore the computed values do not differ too much from the values, that would be computed from the custom training dataset.

Next, each image was normalized using the normalization method described in section 2.8.3.

The above modifications were primarily focused on data enhancements. This improves the overall performance, robustness, and generalization capabilities of the model. The next part focuses on preparing the data for model input.

Since using the EfficientNet model for the initial extraction of features, it is necessary to transform the image into a format, that is compatible with the model’s requirements. This step is essential to ensure that the images can be successfully inputted into the EfficientNet model that represents the initial part of the overall developed model.

Regarding the input shape, each EfficientNet model architecture ranging from B-0 to B-7 is built for different input image shapes. This is due to the fact that each model was designed using compound scaling which scales the depth, width and input image resolution proportionally [24]. Therefore the input image needs to be resized according to the model input shape requirements. The model accepts only square shaped images. However, a typical training image represented by a person bounding box has a rectangular shape. To transform the bounding box into a square shape, a padding was used. In this case, the additional padding layers were added only to the left and right sides of the image to increase the width of the image to reach the desired square shape.

4.2.2 Hyperparameters Tuning

In order to make the most out of the developed model and to achieve the best results possible, tuning of hyperparameters was introduced. The testing was conducted using a systematic and controlled approach to understand the impact of each hyperparameter on the model's performance.

Initially, a model with a basic configuration was created. The base model was chosen to have a set of hyperparameters that are commonly used for similar tasks. The base model configuration consists of the following:

- Feature Extractor: EfficientNet-B0 (pre-trained on ImageNet)
- Optimizer: SGD
- Learning Rate Scheduler: StepLR with step size of 2 and gamma of 10^{-1}
- Initial Learning Rate: 10^{-2}
- Weight Decay: 10^{-4}
- Loss Function: Cross-Entropy Loss
- Batch Size: 32
- Num. of Epochs: 6
- Dropout Rate: 0.5 in the classifier head layers
- Layers in the Classifier head: 9
- Data augmentation technique: One random out of three total for each image

In order to understand the impact of each hyperparameter on the model's performance, a one-at-a-time approach was used. This method involves altering only one hyperparameter at a time, while keeping the others fixed at their default values. The idea behind this approach is to isolate the impact of each hyperparameter, making it easier to interpret the results and identify the most influential hyperparameters as well as finding the best value for each hyperparameter.

The tuning procedure consisted of selecting a hyperparameter to tune. Next, a set of models were initialized, each with a different values of the chosen hyperparameter. These models were then trained and evaluated on the data created by the data creation pipeline.

The first tuning involved finding the most suitable optimizer. The evaluation involved three setups: Adam, Stochastic gradient descent (SGD), and Stochastic gradient descent with a declining learning rate. Each setup was tested with different learning rates ranging from 10^{-2} to 10^{-5}. For the SGD with the declining learning rate, the learning rate value denotes the starting rate. Since Adam optimizer adjusts the learning rate adaptively during training, the learning rate value serves as a baseline from which the optimizer begins the optimization process. To prevent overfitting, a regularization technique, namely weight decay, was used with every setup with a value of 10^{-4}.

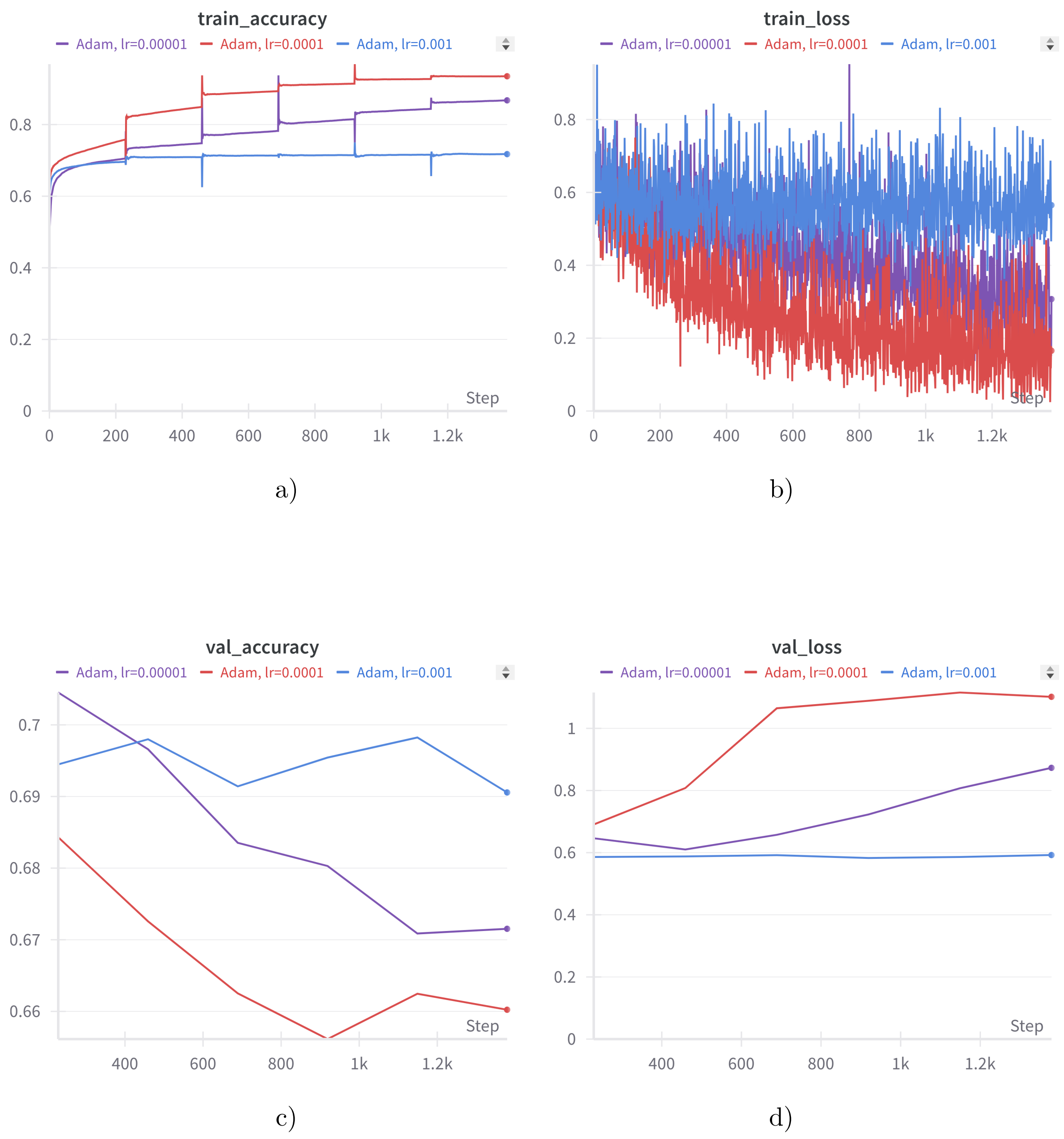


Figure 4.1. The graphs represent the a) training accuracy b) training loss c) validation accuracy d) validation loss of the Adam optimizer over successive training steps. Graphs generated using [46].

Analyzing the results, the Adam optimizer showed signs of overfitting for all learning rates. The model was memorising data rather than learning to generalize from it. This can be seen through increasing performance on the training dataset and a decreasing performance on the validation dataset. The setup with a learning rate of 10^{-2} was too high and lead to a gradient explosion, therefore was omitted from the metrics.

The next setup contained the Stochastic gradient descent optimizer.

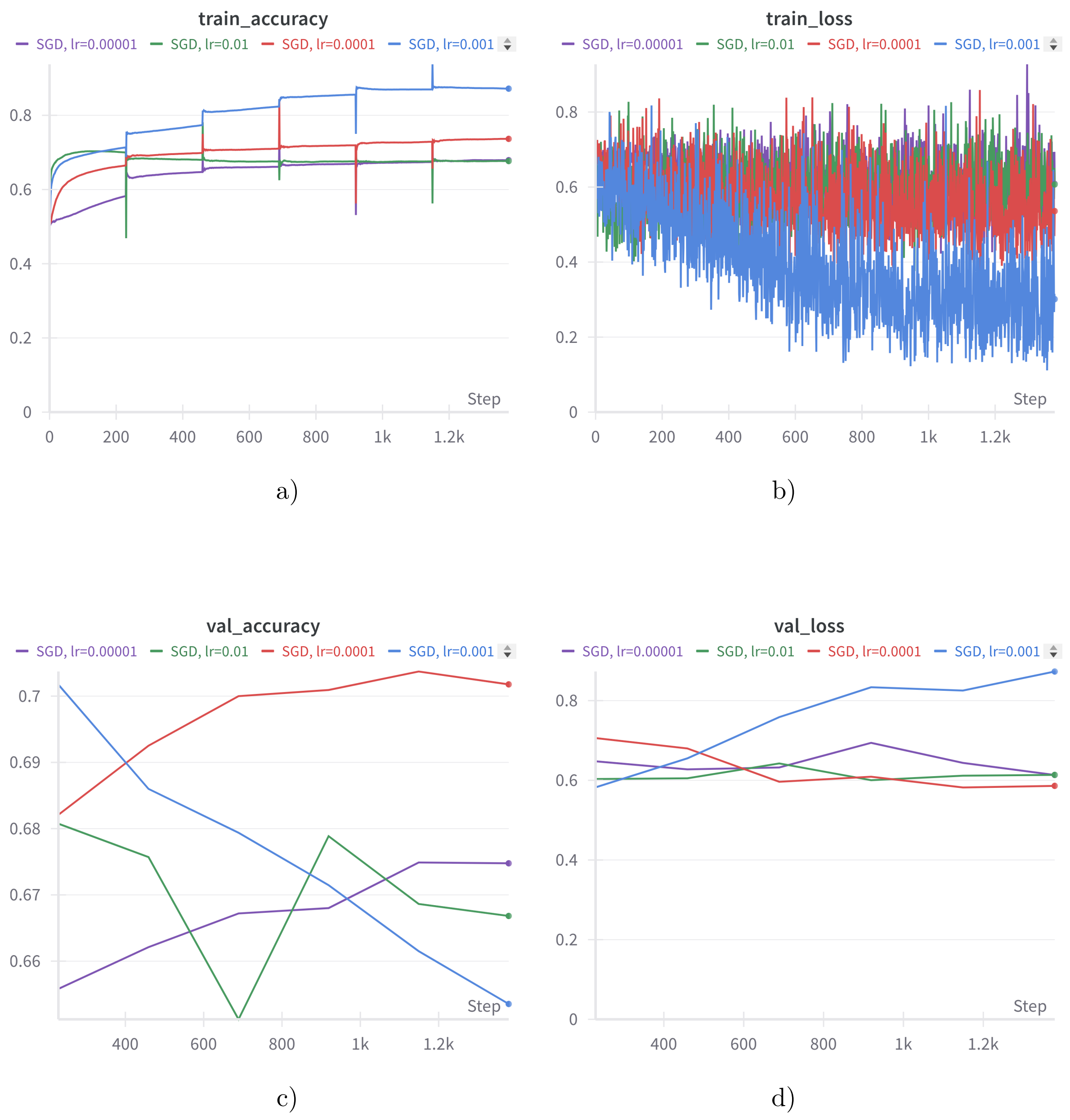
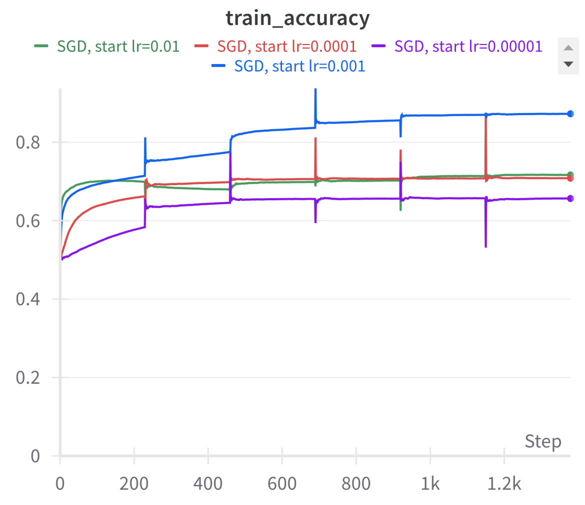


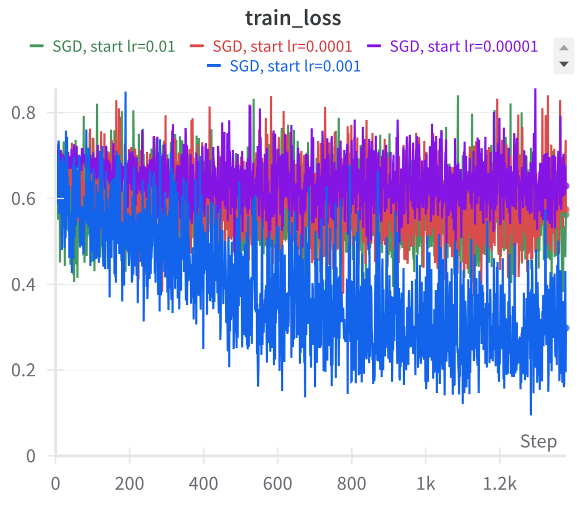
Figure 4.2. The graphs represent the a) training accuracy b) training loss c) validation accuracy d) validation loss of the SGD optimizer over successive training steps. Graphs generated using [46].

The SGD lead to very distinct results. The learning curve on the training dataset shows that the model learned the underlying features in the first and second epoch and became stagnant afterwards. Analyzing the validation results, the model also showed signs of stagnancy, improving the most during the first training epoch.

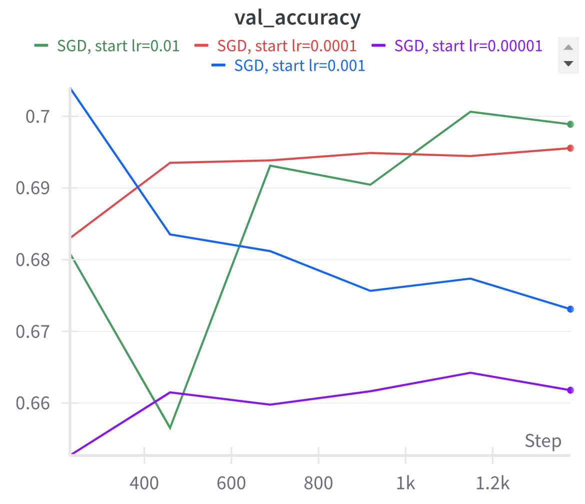
The third setup consisted of SGD together with a learning rate scheduling. The learning rate was decreased every second epoch. This makes it possible to make big steps at the beginning of the training and slowly decrease the size of steps, ensuring that the model converges to a desired local minimum.



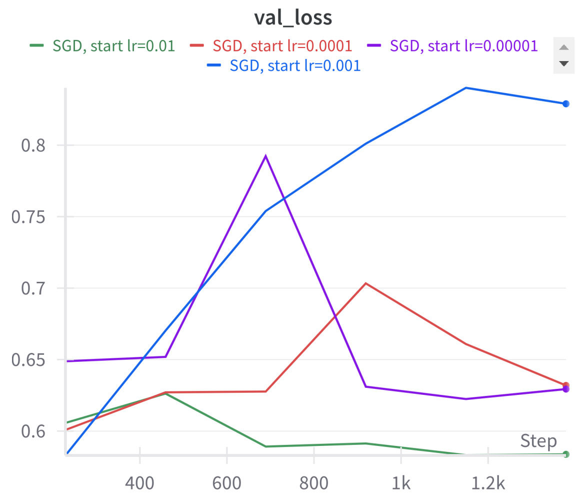
a)



b)



c)



d)

Figure 4.3. The graphs represent the a) training accuracy b) training loss c) validation accuracy d) validation loss of the SGD optimizer with a decreasing learning rate over successive training steps. Graphs generated using [46].

Reflecting on the results, the gradual decrease of learning rate slightly improved the performance. However, is many cases, the validation loss increased. Only the learning rate of value 10^{-2} decreased while also leading to the best accuracy. Comparing the results, the best performing optimizer proved to be the SGD with a decreasing learning rate starting at 10^{-2} reaching an accuracy of 69.24%.

Another important hyperparameter is the batch size. Experimenting with values 32, 64, 128, and 264, with each larger size, the train accuracy improved. On the other hand, the validation accuracy got worse with each larger batch size. This can be due to the fact that a smaller batch size results in a noisier gradient. This acts as a regularization technique, helping the model escape suboptimal local minima and potentially find better solutions. The batch size is often strongly correlated with a learning rate. To reach good performance, a small learning rate typically requires a larger batch size, while a larger learning rate might require smaller batch size.[47] This proved right since the used learning rate was starting at 10^{-2}.

Next step was to test different feature extractors. The base model contained the EfficientNet-b0 model as a feature extractor together with a custom classifier head. Testing each type of the EfficientNet from b0 to b7, the test results showed no signs of improvement. The performance was almost the same for each EfficientNet model. This can be due to a lack of diversity in the dataset or a task simplicity. The EfficientNet-b0 is already a very complex model, increasing the complexity of this feature extractor might not bring any significant improvement.

Data augmentation plays a vital role in enhancing the generalization of a model. The next experiment was conducted for two setups. The first setup randomly chose one out of three possible transformations. Random horizontal flip, random resize and crop, and altering the image brightness, contrast, saturation, and hue. The second setup applied all three transformations on each sample. Applying only one transformation per image resulted in validation accuracy of 69.71% compared to 68.93% when applying all three transformations. Reflecting on the results, applying milder augmentation resulted in better performance.

The base dropout rate was set to 0.5, meaning that half of the connections between the layers are dropped during training. This method was implemented to tackle overfitting and serve as a regularization technique. However, during testing of other hyperparameters, the model often showed signs that are usually associated with underfitting. One of these signs was that the model accuracy started to stagnate after first two epochs. This means that the model struggles to learn from the training data. In order to find out, several smaller dropout values were tested, ranging from 0.2 to 0.5. Reviewing the results, the validation accuracy did improve from 69.64% for dropout rate of 50% to 69.67% for a dropout rate of 30%. The small magnitude of change is mainly due to the fact, that the classifier head, in which the dropout rate was decreased, consisted of only 9 layers, thus the impact on the overall performance was not as significant.

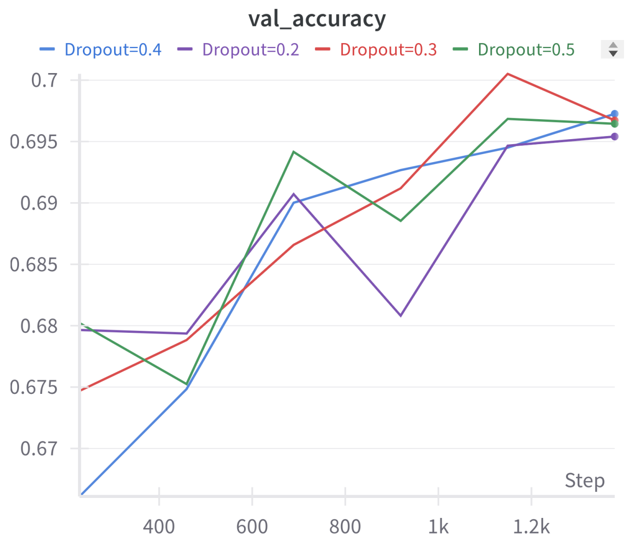
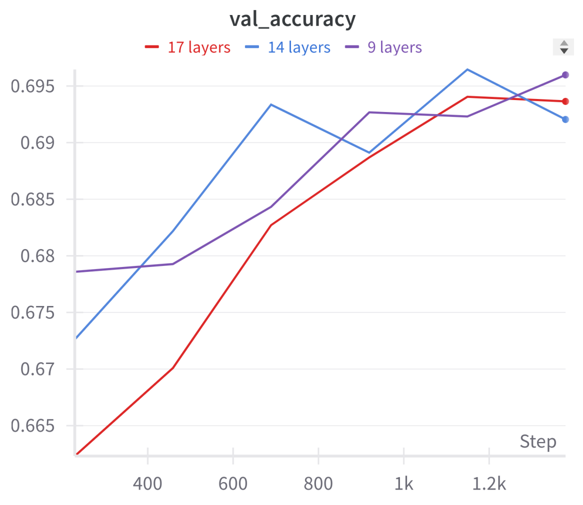
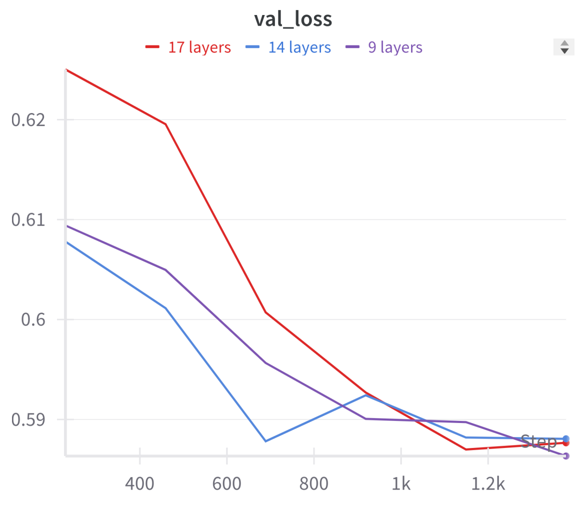


Figure 4.4. Change of dropout rate in the model's classifier head. (Graph generated using [46])

Taking into account the results obtained during the hyperparameters tuning, The model showed signs of underfitting. A possible solution lies in increasing the complexity of the model and thus helping it to learn and generalize better. Therefore the next test focused on developing more complex classifier heads. There were three setups in total. The base classifier head consisted of 9 layers with 3 fully connected layers, each with a preceding dropout and a following activation ReLU function. The second model included a classifier head with 5 fully connected layers. In order to preserve more information from the feature extractor, the average pooling layer kernel size was increased to size 4. The third model additionally contained 6 fully connected layers, totaling to 17 layers in the whole classifier head architecture.



a)



b)

Figure 4.5. The graphs represent the a) validation accuracy b) validation loss of the model with different architecture of the classifier head. Graphs generated using [46].

Reflecting on the results, making the classifier head more complex did not result in any significant improvement. For a more accurate measurement, training for more epochs is needed.

Putting the hyperparameters tuning results together, the final model configuration differs from the initial setup in:

- Number of training epochs that was increased to 12.
- Dropout rate, that was decreased to 0.3.
- Number of layers in the classifier head was increased to 17 layers.

4.3 Performance Evaluation

The final model was trained over 12 epochs. It reached a training accuracy of 72.35% and training loss of 0.553. Performing inference on the validation data, the model reached a validation accuracy of 69.7 and a validation loss of 0.585. The small difference between the values of training and validation accuracy signify, that the model is most likely not overfitting and learned to generalize well on the unseen data.

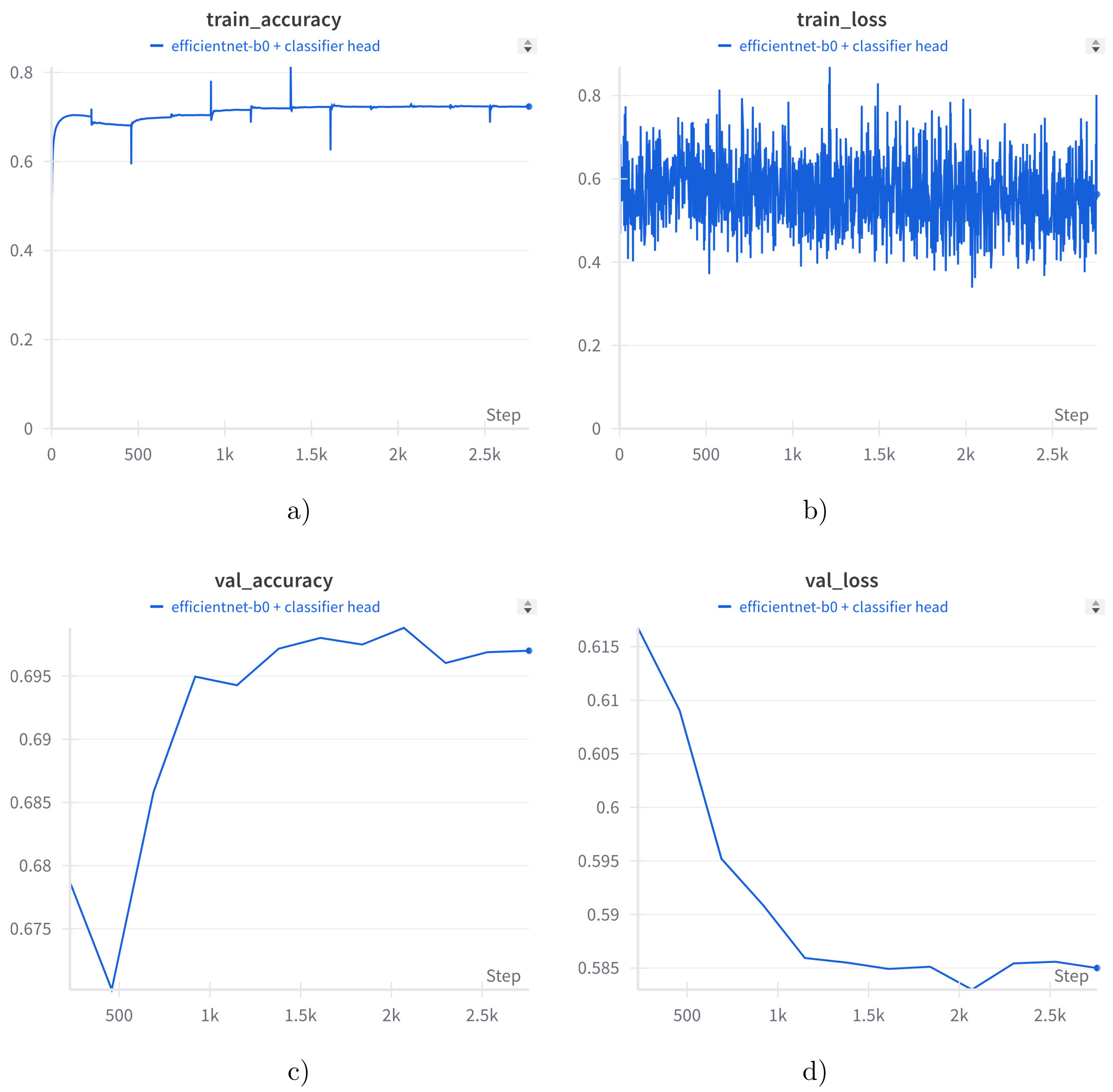


Figure 4.6. The graphs represent the a) training accuracy b) training loss c) validation accuracy d) validation loss during the training process of the final model. (Graphs generated using [46].)

Comparing the results with the initial configuration, the tuning of hyperparameters did not change the final accuracy by a large amount. This can be due to the fact that the initial configuration was already well chosen as the tuning process showed. In the end, only three hyperparameters were changed.

Measuring the model accuracy on the validation data provides a good metric. However, it is important to note that this data was generated using the data creation pipeline with an algorithm for generating both positive and negative data. The method for automatically extracting training data from the BDD100k dataset showed an 96.15% accuracy. This means the training data might contain some portion of wrongly labeled cases.

In order to objectively measure the accuracy of the model to see, how it might perform in the real-world scenarios, the same small manually annotated benchmark dataset from section 3.6.1 was utilized. This establishes a baseline performance metric. Altogether, the dataset comprises of 106 positive, and 106 negative cases, totaling 212 cases. Each positive case captures a moment of approximately 1.3 seconds or 40 frames before a person steps into a road. Each negative case also consists of 40 frames.

The evaluation was performed in the following way. Each positive and negative case, consisting of 40 frames, was processed by extracting 5 representative frames from each case. To ensure the evaluation is consistent across all samples, the predictions made on the positive ground truth data were made on the frames −1, −10, −20, −30, and −40, where the indices indicate frames leading up to the individual stepping into the road. The frame −40 means, that this frame captures the person 40 frames before they entered the road, frame −30 captures the person 30 frames before they entered the road and so on. For negative cases, the naming follows the same rules. The only difference is that it does not represent the number of frames before the individual enters the road, but simply which frame is which from the 40 frame sequence for each negative case. This reduces variability caused by different timings and allows for a fair comparison. This also allows for observation, whether the predictions get better, the closer individuals are to the moment of road entry.

Type of Data	frame −40	frame −30	frame −20	frame −10	frame −1
True positives (TP)	84	84	86	87	84
False negatives (FN)	22	22	20	19	22
True negatives (TN)	72	72	71	63	68
False positives (FP)	33	33	34	42	37
total num. of cases	212	212	212	212	212
Accuracy (%)	73.93	73.93	74.41	71.10	72.04

Table 4.1. Measuring the classifier performance using the ground truth manually annotated data.

The classifier shows a good performance with an average accuracy between 71.10% and 74.41% on the ground truth manually annotated dataset.

The precision of the classifier on the ground truth data is slightly higher than on the data generated by the data creation pipeline that were used for training. Namely, the accuracy on the benchmark dataset is between 71.10% and 74.41%, while the accuracy on the validation data generated by the data creation pipeline is 69.7% as showed on the graph for the validation accuracy in figure 4.6. This suggests, that the classifier generalizes well on unseen data. The overall accuracy reflects the classifier's ability to correctly classify between 71.10% and 74.41% of the cases.

Reflecting on the results in the table 4.1, there seems to be no evidence that the classifier would perform better, the closer the pedestrian is to the road entry. This is most likely due to the reason, that the training data contained all 40 frames of both positive and negative cases. Hence, the classifier should perform uniformly the same as each moment from frame 0 to frame 40 is represented by the same amount of samples in the training dataset. Another reason can be the small size of the validation dataset with only 212 cases. With such a limited number of samples, the dataset may not sufficiently eliminate statistical noise, leading to less reliable and less accurate performance metrics. The differences in results are all within approximately 3%, which is too small to draw meaningful conclusions. A larger validation dataset would be needed for more accurate measurement and to reduce the impact of statistical noise.

Chapter 5
Conclusion

This bachelor's thesis focused on developing an algorithm capable of auto-labeling moments in which pedestrians enter the road. This was further utilized in the data creation pipeline capable of automatically extracting cases in which pedestrians enter the road. To leverage the auto-labeling algorithm, a custom classifier was developed and trained leveraging the data creation pipeline. Both the performance of the auto-labeling algorithm as well as of the model were then evaluated on a manually annotated ground truth dataset.

Regarding the development of the auto-labeling algorithm, two major methods were tested. The first method used a predefined trapezoidal area for classifying the road area. The second method leveraged the state-of-the-art semantic segmentation model named Segformer. The first method demonstrated a poor performance with recall of 22.43%. However, the second method showcased a significant improvement, reaching the precision of 96.15% and recall of 70.75%. This demonstrates that the auto-labelling algorithm achieves good results and can serve as the baseline for further development.

The developed classifier, trained on the data generated by the data creation pipeline, focused on classifying whether a given person is/is not going to enter the road in the next 40 frames. It reached an overall accuracy that ranged between 71.10% and 74.41% (depending on the index of the frame for each case) on the manually annotated ground truth dataset, compared to an accuracy of 69.70% reached on the validation data during training. This demonstrated classifier's ability to generalize well on unseen data.

Looking ahead, several improvements and extensions can be made to enhance the performance of the auto-labeling algorithm. The following can serve as suggestions for future work. Since the data creation pipeline extracts only bounding boxes of individuals as training data, the model can only extract a limited amount of features from the image. By enlarging the bounding box by a few pixels on all four sides, the model might be able to extract more relevant features and thus improve its classification performance. During the experiments, the ByteTrack tracking algorithm showed a sub-optimal performance during occlusion. The ID switches occurred quite often in crowded scenes. Experimenting with different algorithms that also utilize appearance-based features could enhance the tracking performance. These features can be particularly beneficial during the reassociation stage in cases, where individuals are close to or are occluding each other. As the experiments in section 3.6 showed, the quality of road area selection significantly influences the performance of the auto-labeling algorithm. The current semantic segmentation method showed a mediocre performance along the edges of a road. Since the current algorithm relies heavily on the transition that occurs at the edge of the road, being able to find the road edge with better precision could result in an increase of the algorithm performance.

References

[1] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets Robotics: The KITTI Dataset. *International Journal of Robotics Research (IJRR)*. 2013.

[2] 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*. 2020.

[3] Yali Amit, Pedro Felzenszwalb, and Ross Girshick. *Object Detection*. In: Katsushi Ikeuchi, eds. *Computer Vision: A Reference Guide*. Cham: Springer International Publishing, 2021. 875–883. ISBN 978-3-030-63416-2.

[4] Adam Paszke, Sam Gross, Francisco Massa, and Lerer. *PyTorch: An Imperative Style, High-Performance Deep Learning Library*. In: H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlche-Buc, E. Fox, and R. Garnett, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2019.

[5] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollar, and C. Lawrence Zitnick. *Microsoft COCO: Common Objects in Context*. In: David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, eds. *Computer Vision – ECCV 2014*. Cham: Springer International Publishing, 2014. 740–755. ISBN 978-3-319-10602-1.

[6] Yuxin Wu, Alexander Kirillov, Francisco Massa, Wan-Yen Lo, and Ross Girshick. *Detectron2*.
`https://github.com/facebookresearch/detectron2`. 2019.

[7] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. *Mask R-CNN*. In: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. 2017.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[9] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. *Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks*. In: C. Cortes, N. Lawrence, D. Lee, M. Sugiyama, and R. Garnett, eds. *Advances in Neural Information Processing Systems*. Curran Associates, Inc., 2015.

[10] Mardhiyah Md Jan, Nasharuddin Zainal, and Shahrizan Jamaludin. Region of interest-based image retrieval techniques: a review. *IAES International Journal of Artificial Intelligence*. 2020, 9 (3), 520.

[11] Peifeng Su, Jorma Joutsensaari, Lubna Dada, Martha Arbayani Zaidan, Tuomo Nieminen, Xinyang Li, Yusheng Wu, Stefano Decesari, Sasu Tarkoma, Tuukka Petäjä, Markku Kulmala, and Petri Pellikka. New particle formation event detection with Mask R-CNN. *Atmospheric Chemistry and Physics*. 2022, 22 1293-1309. DOI 10.5194/acp-22-1293-2022.

[12] Jonathan Hui. *Understanding Feature Pyramid Networks for object detection (FPN) [Online] [Cit. 1.5.2024]* . 2018.
https://jonathan-hui.medium.com/understanding-feature-pyramid-networks-for-object-detection-fpn-45b227b9106c.

[13] Dive deepintodeep learning. *Residual Networks (ResNet) and ResNeXt [Online] [Cit. 30.3.2024]* .
https://d2l.ai/chapter_convolutional-modern/resnet.html.

[14] Sasha Targ, Diogo Almeida, and Kevin Lyman. *Resnet in Resnet: Generalizing Residual Architectures*. 2016.

[15] Zahra Soleimanitaleb, Mohammad Ali Keyvanrad, and Ali Jafari. *Object Tracking Methods:A Review*. In: *2019 9th International Conference on Computer and Knowledge Engineering (ICCKE)*. 2019. 282-288.

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

[17] Yifu Zhang, Peize Sun, Yi Jiang, Dongdong Yu, Fucheng Weng, Zehuan Yuan, Ping Luo, Wenyu Liu, and Xinggang Wang. *ByteTrack: Multi-Object Tracking by Associating Every Detection Box*. 2022.

[18] Jeremy Jordan. *An overview of semantic image segmentation. [Online] [Cit. 1.5.2024]* . 2018.
https://www.jeremyjordan.me/semantic-segmentation/.

[19] Enze Xie, Wenhai Wang, Zhiding Yu, Anima Anandkumar, Jose M Alvarez, and Ping Luo. SegFormer: Simple and efficient design for semantic segmentation with transformers. *Advances in neural information processing systems*. 2021, 34 12077–12090.

[20] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale. *CoRR*. 2020, abs/2010.11929

[21] Jurgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*. 2015, 61 85–117.

[22] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. *Amc: Automl for model compression and acceleration on mobile devices*. In: *Proceedings of the European conference on computer vision (ECCV)*. 2018. 784–800.

[23] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. *Mobilenetv2: Inverted residuals and linear bottlenecks*. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2018. 4510–4520.

[24] Mingxing Tan, and Quoc V. Le. *EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks*. 2020.

[25] Jason Brownlee. *How to Choose Loss Functions When Training Deep Learning Neural Networks [Online] [Cit. 12.5.2024]* . 2020.
https://machinelearningmastery.com/how-to-choose-loss-functions-when-training-deep-learning-neural-networks/.

[26] Davide Giordano. *7 tips to choose the best optimizer [Online] [Cit. 13.5.2024]* . 2020.

https://towardsdatascience.com/7-tips-to-choose-the-best-optimizer-47bb9c1219e.

[27] Wikipedia contributors. *Stochastic gradient descent [Online] [Cit. 13.5.2024]* . 2023.
https://en.wikipedia.org/wiki/Stochastic_gradient_descent.

[28] Mohit Mishra. *Stochastic Gradient Descent: A Basic Explanation [Online] [Cit. 13.5.2024]* . 2023.
https://mohitmishra786687.medium.com/stochastic-gradient-descent-a-basic-explanation-cbddc63f08e0.

[29] Harsh Yadav. *Dropout in Neural Networks [Online] [Cit. 12.5.2024]* . 2022.
https://builtin.com/machine-learning/fully-connected-layer.

[30] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*. 2014, 15 (56), 1929–1958.

[31] Ajitesh Kumar. *Weight Decay in Machine Learning: Concepts [Online] [Cit. 13.5.2024]* . 2022.
https://vitalflux.com/weight-decay-in-machine-learning-concepts/.

[32] Shreenidhi Sudhakar. *Histogram Equalization [Online] [Cit. 14.5.2024]* . 2017.
https://towardsdatascience.com/histogram-equalization-5d1013626e64.

[33] Aarzoo Garg. *Standardization v/s Normalization [Online] [Cit. 14.5.2024]* . 2023.
https://medium.com/@meritshot/standardization-v-s-normalization-6f93225fbd84.

[34] Aniruddha Bhandari. *Feature Scaling: Engineering, Normalization, and Standardization [Online] [Cit. 14.5.2024]* . 2024.
https://www.analyticsvidhya.com/blog/2020/04/feature-scaling-machine-learning-normalization-standardization/.

[35] Ashish Sanjay Raut. *Padding in Neural Networks: Why and How? [Online] [Cit. 14.5.2024]* . 2023.
https://blog.gopenai.com/padding-in-neural-networks-why-and-how-b076ab0a4fc2.

[36] Vivek Chaudhary. *What is “padding” in Convolutional Neural Network? [Online] [Cit. 14.5.2024]* . 2023.
https://www.almabetter.com/bytes/articles/what-is-padding-in-convolutional-neural-network.

[37] G. Bradski. The OpenCV Library. *Dr. Dobb’s Journal of Software Tools*. 2000.

[38] MMSegmentation Contributors. *MMSegmentation: OpenMMLab Semantic Segmentation Toolbox and Benchmark*.
https://github.com/open-mmlab/mmsegmentation. 2020.

[39] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The Cityscapes Dataset for Semantic Urban Scene Understanding. *CoRR*. 2016, abs/1604.01685

[40] Wikipedia contributors. *k-d tree [Online] [Cit. 12.5.2024]* . 2021.
https://en.wikipedia.org/wiki/K-d_tree.

[41] Andy B Yoo, Morris A Jette, and Mark Grondona. *Slurm: Simple linux utility for resource management*. In: *Workshop on job scheduling strategies for parallel processing*. 2003. 44–60.

[42] Niklas Donges. *What Is Transfer Learning? Exploring the Popular Deep Learning Approach. [Online] [Cit. 12.5.2024]* . 2022.
`https://towardsdatascience.com/the-basics-of-neural-networks-neural-network-series-part-1-4419e343b2b`.

[43] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. *Imagenet: A large-scale hierarchical image database.* In: *2009 IEEE conference on computer vision and pattern recognition*. 2009. 248–255.

[44] Abhishek Kumar Pandey. *Adaptive Average Pooling Layer [Online] [Cit. 12.5.2024]* . 2024.
`https://medium.com/@akp83540/adaptive-average-pooling-layer-cb438d029022`.

[45] Prudhviraju Srivatsavaya. *Flatten Layer — Implementation, Advantage and Disadvantages [Online] [Cit. 12.5.2024]* . 2023.
`https://medium.com/@prudhviraju.srivatsavaya/flatten-layer-implementation-advantage-and-disadvantages-0f8c4ecf5ac5`.

[46] Lukas Biewald. *Experiment Tracking with Weights and Biases [Online] [Cit. 20.5.2024]* . 2020.
`https://www.wandb.com/`. Software available from wandb.com.

[47] Priya Goyal, Piotr Dollár, Ross Girshick, Pieter Noordhuis, Lukasz Wesolowski, Aapo Kyrola, Andrew Tulloch, Yangqing Jia, and Kaiming He. *Accurate, Large Minibatch SGD: Training ImageNet in 1 Hour*. 2018.