FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Assignment of master's thesis

Title:	Improving Pedestrian Detector via Occlusion Prediction
Student:	Bc. Martin Koucký
Supervisor:	Ing. Filip Naiser
Study program:	Informatics
Branch / specialization:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	until the end of summer semester 2022/2023

Instructions

A student is going to deal with pedestrian occlusion detection. Occlusion is a phenomenon in computer vision that brings inaccuracy to the detection process. Since the accuracy of detection affects tracking and all the other high-level tasks, we think it could have a significant impact on tracking and high-level tasks built on top of that.

At first, he performs a literature review on this topic. Then, he will design a dataset format. We expect most of the training data to be harnessed from our internal datasets. If needed, the student will design heuristics to choose frames for annotation with a high chance of occlusion to enlarge his dataset. Next, the student will design and train a neural network predicting whether, in a given image crop, there is an occlusion happening. He will collaborate with the tracking team and together evaluate the impact of occlusion knowledge on tracking performance.

Electronically approved by Ing. Karel Klouda, Ph.D. on 12 October 2021 in Prague.

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Improving Pedestrian Detector via Occlusion Prediction

Bc. Martin Koucký

Department of Applied Mathematics
Supervisor: Ing. Filip Naiser

May 2, 2022

Acknowledgements

I would like to thank my supervisor, Ing. Filip Naiser, for his advice and guidance throughout the process of developing and writing this thesis.

Declaration

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

I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. I further declare that I have concluded an agreement with the Czech Technical University in Prague, on the basis of which the Czech Technical University in Prague has waived its right to conclude a license agreement on the utilization of this thesis as a school work under the provisions of Article 60 (1) of the Act. This fact shall not affect the provisions of Article 47b of the Act No. 111/1998 Coll., the Higher Education Act, as amended.

In Prag on May 2, 2022 . . . . . .

Czech Technical University in Prague
Faculty of Information Technology

Citation of this thesis

Koucký, Martin. *Improving Pedestrian Detector via Occlusion Prediction.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2022.

Abstract

Pedestrian occlusion is a well-known problem in camera vision detection and tracking of pedestrians. We aim to improve the existing tracking system developed by iC Systems.ai by adding a new predicted feature – the probability of detection being a pair of pedestrians.

The main contribution of this work is the description of the dataset creation process and the solutions to the encountered problems. Our approach and the resulting model lead to promising results when being evaluated on real-life data.

The data in our datasets come from iC Systems.ai cameras from several shopping centers, where the existing tracking system is deployed.

Keywords CNN, camera vision, pedestrian tracking, occlusion detection

Abstrakt

Okluze chodců je dobře známým problémem při detekci a sledování chodců kamerovým viděním. Naším cílem je vylepšit stávající systém sledování vyvinutý společností iC Systems.ai přidáním nové predikce pravděpodobnosti, že detekce je ve skutečnosti dvojice chodců.

Hlavním přínosem této práce je popis procesu tvorby datové sady a řešení vzniklých problémů. Náš přístup a výsledný model vedou ke slibným výsledkům při vyhodnocování na reálných datech.

Data v našich datasetech pocházejí z kamer iC Systems.ai z několika obchodních center, kde je nasazen stávající sledovací systém.

Klíčová slova CNN, kamerová vize, sledování chodců, detekce okluze

Contents

List of Figures

List of Tables

Introduction

The problem of occlusion in camera vision is well known but rarely addressed as explained in the chapter 1. Rather, occlusion is often the explaining factor behind the error in tasks such as counting. In this work, we aim to improve the existing tracking system of iC Systems.ai in the pedestrian counting task by adding a pair occlusion classifier. The counting task is useful for example during the pandemic, where the number of pedestrians in a shopping center has to be under some threshold. It consists of first detecting and tracking pedestrians and then counting them when they reach a predefined counting zone.

We use data from iC Systems.ai cameras which are placed on the ceiling looking directly down in many shopping centers. The cameras use limited lightweight hardware so our tracking system has to be lightweight and fast as well. The data are manually annotated by drawing a rectangle representing a detection over each pedestrian in the image. The dataset creation process is described in the chapter 3.

We use a Convolutional neural network described in the chapter 2 as well as a multi-task learning system which we call hydra also described in the chapter 2.

During the experiments described in the chapter 4 we encountered several significant problems. First, our dataset created from the manual annotations doesn't currently have enough samples (the number of class pair samples mostly fluctuates around a thousand depending on the dataset). Further, the dataset is heavily imbalanced in favor of the class with single pedestrians (more than 10 to 1 ratio with most settings). And finally, a big part of the samples in the created dataset had unclean, often mislabeled data.

We solve some of these problems by manually cleaning the data, creating heuristics so that more annotations lead to the class pair (by sending proportionally more images possibly containing pairs to the annotators), and by adding augmentations. The final results on the validation dataset show a promising precision of around 0.70, recall of around 0.78, and accuracy of

around 81%. In the final evaluation, the pairs classifier is applied to tracking in the counting task to determine whether a detection of a pedestrian is instead a detection of two pedestrians in which case, another detection is created and two pedestrians are counted. This led to a decrease in the counting error from 117 to 111 errors on 2251 passages. Note that this success may be inflated because of the negation of false positives and false negatives. The result is not yet applicable, but it is promising and shows that further research and development may significantly improve the tracking.

Goals

Our goal in the theoretical part of the thesis is to research related works about occlusion detection in camera vision, particularly in the domain of pedestrian detection. We also aim to explain how the CNN used in the practical part of this thesis works.

In the practical part of the thesis, our goal is to create suitable datasets from the data harnessed from iC Systems.ai cameras. Then we train a pairs occlusion binary classifier and use several data engineering methods to overcome problems with the datasets and achieve satisfiable results on the validation dataset such as heuristics to enlarge our dataset or manual data cleaning. Our final goal is to evaluate the performance of the existing tracking system with the addition of the new pairs classifier in the counting task and discuss the results.

Chapter 1

Related work

In this chapter, we explore work related to ours. This gives us more insight and understanding of our problem as well as provides inspiration for new ideas we could use to solve our task.

Research about the problem of occlusion is often tied to the problem of counting. Carlos Arteta et. al. mention in their work[1] that methods of counting generally fall into two categories. Density based counting and detection based counting. While density based counting offers better results in crowded environments with lots of overlapping instances. It is based on probability and rarely offers more information than the estimated count. As our framework requires additional information, we had to disregard most of the works dealing with occlusion and focus solely on those that count using detections.

When searching for papers related to our problem we found out that not many papers deal with the problem of occlusion. The problem of occlusion is often blamed for inaccuracies, but it is rarely directly addressed. Aside from the works described below, we examined several others [2][3][4][5][6][7][8] but ultimately did not find their contributions useful for our task or they were too similar in concept to those described below.

1.1 A novel method for detecting and counting overlapping tracks in SSNTD by image processing techniques

Detecting objects for the purpose of counting is the focus of research in many different fields of science. N. Ab Azar et. al. [1] deals with this problem in the domain of round particles counting (eg. cells counting).

Their novel approach tries to solve the problem of overlapping particles using junction points. Given two overlapping circles, junction points are the

[1] ABAZAR201636

points of their intersection, when we consider only the circle circumferences. This means that two circles, that are both visible, can have at maximum 2 junction points.

Given the count of all junction points (n_{jp}), the total number of particles (n_t) can be calculated with the following formula.

$$n_t = \frac{n_{jp}}{2} + 1,$$

The junction points are determined by assigning local neighborhood values to all pixels and choosing the ones with the highest value of a heuristic equation for determining junction points.

1.1.1 Conclusion in relation to our work

Although the counting accuracy they were able to achieve was very high (97% on pair overlap), the method could not be used for our problem. Our domain of pedestrians has difficulties when we look for a heuristic approach. Shapes of pedestrians don't have the same geometric properties, so such a simple heuristic with junction points can't be applied.

1.2 Learning to detect partially overlapping instances

To detect objects in images Carlos Arteta et. al.[1] divide the area into non-overlapping regions and then classify them. Their addition is the novel classification system, where regions can be classified as tuples or single objects. This, in comparison to the default idea of classifying each item as either an object or nothing, has the potential to solve occlusion. This is because in theory all the occluded objects are just classified as "tuples".

They conduct their experiments on the UCSD pedestrian dataset. In comparison to ours, this dataset has a less varied view angle of the pedestrians. The UCSD images are taken from further away and have a less varied view angle of the pedestrians. The images have a bigger scope and more people present in one image than the images from our datasets.

Their method learns to assign different integer sizes to regions containing objects based on density maps. Then it predicts the locations of the instances of an object inside the region. The later location prediction being easier because the method already has the total count of objects.

The addition of the novel classification system to their previous detection method resulted in a 2 % increase in the counting accuracy on the UCSD pedestrian dataset for a maximum of 0.895 F_1 score accuracy.

1.2.1 Conclusion in relation to our work

Unfortunately, our detector needs to be more lightweight and also work within the hydra framework so their solution can't be fully applied to our problem. But we experiment with the same type of classification, where we first detect a pedestrian and then try to determine whether the detection is actually a single person or a pair of (occluding) people. The later classification being done in one of the heads of our hydra architecture.

We also can't apply their density-based method of estimating integer counts of objects inside groups (regions), because our dataset of images taken with a top-down camera has varied magnitude and density of pedestrian groups, because of the varied camera angles. Thus, groups have different densities despite having the same count of pedestrians.

Figure 1.1: visualisation of the perspective problem on theoretical (manually drawn) density maps with top down camera image

green = pedestrians that are farther and appear smaller

1.2.2 Related works

As the aforementioned work[1] was closest in its solution to our problem, we decided to examine all the works citing it. The interesting ones are summarized in this section. Note that there is a year written next to each of the examined works because we believe that the novelty of the research is crucial in quick understanding of the relevancy of examined works citing the aforementioned work from 2013. Also, note that the works are presented in order of how many times they were cited on google scholar from most to least.

Reccurent instance segmentation, [2016]

Bernardino Romera-Paredes et. al.[9] focus on the problem of instance segmentation. They try to create an algorithm that approaches this problem sequentially, examining one object after another just like a human would when counting multiple objects in an image. They also use a recurrent neural network, which means that they are able to reevaluate and recount the instances

of the objects. This is important for occlusion detection as their architecture allows them to revisit instances that the segmentation previously classified as single detections and reclassify them as multiples. The recurrent neural networks are used because of their spatial memory ability meaning that they can keep the current state of instances in an internal memory similarly to human counting.

In their architecture, the examined image is an input to a fully convolutional network. The output of the FCN is then the input to the recurrent neural network. The first iteration in the sequence finds the segmentation of one instance in the image as well as the confidence score of the prediction. The inner state of the RNN is updated to account for the newly found instance. This is repeated until the stopping condition based on the threshold of confidence score is met and ideally, all the instances have been correctly segmented.

They show that their approach alone achieves results comparable to state-of-the-art methods for image segmentation. Further, when conditional random field is added in post-processing to supplement the low resolution representation in the ConvLSTM, the results outperform their state-of-the-art competitors.

Our company approach prevents us from using RNNs because our architecture has to be both more lightweight and more importantly has to work within the wider framework of the hydra architecture. However, we have a similar approach in a human-like detection of instances, where we first detect the instance and then determine whether the detection is a single pedestrian or multiple occluded pedestrians. Note that only the last part of this process is the focus of this work.

Interactive object counting, [2014]

Carlos Arteta et. al.[10] focus on object counting in the field of biomedical image analysis. They develop an interactive system, which means that the user can add his own annotations to a part of the image and the system annotates the rest of the image automatically based on the user's input. If dissatisfied with the result, the user can annotate the wrong parts and launch the automatic annotation again. If the user is satisfied, the system returns the final count of instances.

The system is based around density estimation as research shows that for the ultimate goal of counting instances this density-based approach is the most accurate. In contrast to us, they use a density-based model instead of a detection-based model. Similarly to us, they first find regions but contrary to us they integrate over the regions to get the count of instances inside them.

Extremely overlapping vehicle counting, [2016]

Ricardo Guerrero-Gómez-Olmedo et. al.[11] explore the problem of counting vehicles in a traffic congestion situation where a huge amount of occlusion occurs. They introduce a novel dataset with 1200 annotated samples of overlapping vehicles as well as a new GAME (Grid Average Mean absolute Error) metric. GAME is a modification of MAE (Mean Absolute Error) that improves the precision of the metric for the counting task by dividing the full image into regions. The MAE is then measured in each region separately which prevents compensation of errors. For example when nothing is present and it is falsely counted as an instance (+1) and sometimes the instance is not counted (+0), which gives the right count but isn't correct internally.

Beyond counting: Comparisons of density maps for crowd analysis tasks—counting, detection, and tracking, [2018]

Di Kang et. al.[12] compare the density maps of crowds produced with different methods for crowd analysis tasks. The examined tasks are counting, detection and tracking. Generally, it can be said that the methods that performed the best in counting tasks such as density maps created with fully-convolutional neural networks (such as MCNN) have performed poorly in localization-based tasks. On the other hand, dense pixel-prediction using CNN-pixel led to the best results in localization tasks but was worse in the counting task.

Graphical model for joint segmentation and tracking of multiple dividing cells, [2016]

Focusing on the problem of cell tracking, Martin Schiegg et. al.[13] aim to prevent the propagation of segmentation error into tracking. Their probabilistic model combines more segmentation hypotheses in time into the final tracking. The process works as follows. First, the raw data from t_i (first timestep) and t_{i+1} are segmented separately. Then the segmentations are merged into bigger segments forming segmentation hypotheses (t_i and t_{i+1} are still handled separately). Finally, a graphical model is formed from the two hypotheses where conflicts are resolved and the output is tracking. With this approach, they are able to achieve better than state-of-the-art results on two challenging biology datasets (cells and embryos tracking).

Where are the blobs: Counting by Localization with point supervision, [2018]

Issam H. Laradji et. al.[14] try to solve the counting problem using a detection-based method that does not need to learn the sizes and shapes of the objects thus performing better for the easier task of counting. Their training dataset

of point-only annotations is input to a fully convolutional neural network. Their novel loss function encourages the network to output a single blob for an object annotated with point-only annotations. The loss function consists of four terms, where the terms image-level and point-level loss encourage the network to output segmentation labels for all pixels in the image. The other two terms, split-level and false positive loss enforce the network to predict a unique blob for each unique object and disregard blobs without unique object instances assigned to them. They show that with their approach they are able to outperform current state-of-the-art methods in counting on datasets where a large amount of occlusion is present.

Small instance detection by integer programming on object density maps, [2015]

Zheng Ma et. al.[15] propose a method to detect object instances with small detail such as flocks of birds, biology cells, or crowds of pedestrians in small resolution images. They use center-point annotations, which are then transformed into approximate densities to train their model to estimate the density map of an image. Those annotations are used because they are less labour intensive.

Their approach works in the following way. First, they estimate the density map of the whole image and divide the map into regions. Then a sliding window goes through each of the ROI (regions of interest) and calculates the count of objects. Finally, 2D integer programming is used to find the exact locations of all the object instances from the ROI counts based on the density map. The bounding box of the object instances is then determined with the use of the density map.

Although their approach doesn't specifically deal with the problem of occlusion, it nonetheless achieves state-of-the-art results on some challenging datasets containing occlusion such as the UCSD pedestrians dataset. They even achieve the best F1 score out of all the other examined methods on the UCSD dataset.

Pedestrian detection in crowded scenes via scale and occlusion analysis, [2016]

Lu Wang et. al.[16] propose a novel detection algorithm with occlusion analysis, where the main idea is that the detector of pedestrians will work better when the scale of the pedestrians is known. They try to estimate this scale with neighbors of the pedestrian in question. If the detection is consistent with its neighborhood then its score is not penalized, if it is inconsistent then the score is penalized. This favours detections that are similar in scale to their closest detections, which simulates behaviour in reality, where farther objects

are similar to other far objects that are in their proximity in the image due to perspective.

After scale estimation is done, occlusion analysis via root filter decomposition follows for the purpose of discouraging false positive detections. The fully-body root filter is decomposed into blocks which contain deformable body parts. Then occluding pair detections are classified as either an occluder (the one who occludes) or occludee (the one who is being occluded). Then the confidence score of the occluded detection is recalculated based on its most occluding (highest percentage overlap) occluder. The final confidence score of a partly visible detection is the minimum between the confidence score of its parts and the confidence score of its occludee detection.

The algorithm proposed in the examined work achieves better results than state-of-the-art models on datasets of crowds of pedestrians with a significant amount of occlusion. Note that in our algorithm for dataset creation we also retain information about the detections being either occluded or occluding (or both). As of writing this work, we do not use this gained information, but it is part of our datasets and may be used in the future.

1.2.2.1 Conclusion to the related works

Many examined works related to the aforementioned work [1] had a similar approach of finding interesting regions thus dividing the space and then trying to classify the smaller regions. Interesting note is that similarly to our approach (dividing the image into regions), even the examined methods whose ultimate goal is counting using density, applied similar *divide and conquer* based strategies. We did not directly use any of the methods from the researched works, but we learnt that other works about the topic of detection try to use similar solutions to the problem of occlusion as we do.

1.2.3 Conclusion

Ultimately we couldn't directly use any of the methods from the examined papers, because the model had to fit into our hydra pipeline, which trains multiple heads for multiple tasks at the same time so our task had to have similar to the other tasks. Our architecture is described in more detail in the next chapter. However we gained valuable insights about the challenges as well as inspiration and confirmation that our method of divide and conquer into specific regions is widely used in different forms.

Chapter 2

Methods

In this chapter, we introduce and describe the methods used later in Chapter 4. The main part of this chapter explains how the convolutional neural network works in general and then describes the structure of our CNN. The other section describes our hydra architecture or as it's also called the multi-tasks learning architecture wherein a multitude of heads can be attached to one backbone in a CNN and several tasks learn together.

2.1 CNN

Over the last few decades, Convolutional neural networks have become one of the most popular solutions among deep neural networks due to their ability to handle huge amount of data and have excellent performance in machine learning problems. According to Saad Albawi et. al.[17] they are best used in applications that deal with image data, computer vision, and NLP.

The most beneficial feature of a CNN is its ability to reduce the number of parameters in a neural network. Problems that are to be solved with a CNN should ideally have spatially independent features. For example, in image classification, a convolutional network recognizing faces has only the concern of finding faces and does not need to concern itself with the positions of the faces in the image (although the positional information is still retained in a simplified form). Another advantage of a CNN is that it can obtain abstract features in its deeper layers. For example in image classification, this means learning edges and simple shapes in its first two layers and more complex and abstract constructs as the input is propagated deeper.

2.1.1 Layers

Convolutional neural networks comprise of many different layers. The most important are the convolutional and the pooling layers. In this subsection, we

explain those as well as other layers. Finally, we describe what layers are used in our CNN.

Convolution

We will explain the convolution layer with an example. Assume an input in the form of an image with a height of 32 pixels, width of 32 pixels, and depth of 3 for the RGB channels. To connect this input to one neuron in the hidden layer we would need $32 \times 32 \times 3$ weight connections. If we connect another neuron then the cost increases to $32 \times 32 \times 3 \times 2 = 6000$ weight parameters. However, two neurons might not be enough for any meaningful image classification task so we might want to make the network more efficient by increasing the number of neurons in the hidden layer to match the weight and height of the image. This means the complexity increases to $(32 \times 32 \times 3) \times (32 \times 32) = 3,145,728$ connections. That is where convolution comes in to decrease this complexity.

It has been found[17], that to make the process more efficient, we can look for local regions and connect only those regions instead of a full picture. So each neuron in a hidden layer only gets part of the information from the previous layer. For example, instead of 32×32 it would be connected to a smaller area of 5×5 pixels reducing the previous complexity calculation to $(5 \times 5 \times 3) \times (32 \times 32) = 76,800$ connections which is significantly less than the previous $3,145,728$

The 5×5 matrices move through the input image matrix similarly to a sliding window. This sliding process is called convolution giving the name to the CNN. The matrices are called filters because they work similarly to classic image processing filters like a sharpening filter or a gaussian blur filter. Contrary to the simple graphical filters, the ones in CNNs are able to learn high level features in the deeper layers. An example of a 3×3 edge detection filter can be seen in figure 2.1.

Figure 2.1: edge detection filter matrix and its corresponding visualisation

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

(a) edge detection filter matrix

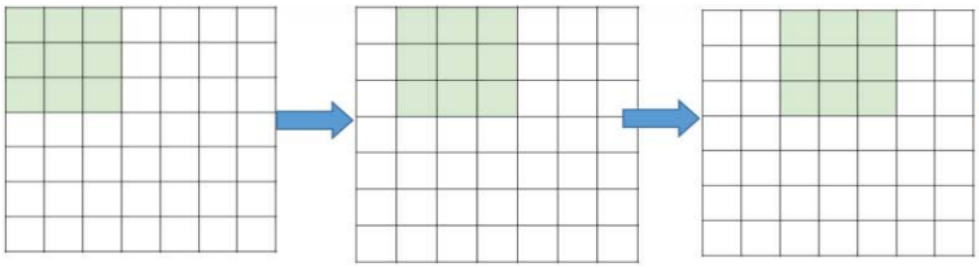
(b) visualisation of the learned edge detection, image source: [17]

Stride

Notice that the sliding window matrix has a significant overlap of examined pixels. To further reduce the parameters, we can use stride to manipulate how

big this overlap between neighbouring matrices is. Stride is essentially how much we move the sliding window. As can be seen in figure 2.2 a 7×7 input with a 3×3 filter and stride 1 will produce a $\frac{7-(3-1)}{1} \times \frac{7-(3-1)}{1} = 5 \times 5$ output. Notice that for example, the first and second matrices have an overlap of 6 (or $\frac{2}{3}$ of the window). If we set stride to 2, the window will move by 2 pixels to the right instead of 1, and the overlap of the first and second window will only be 3 and the output will be reduced to a 3×3 matrix.

Figure 2.2: moving window filter visualisation, image source: [17]



Padding

Notice that not all information is represented with the same attention. The pixels in an image input close to the border will not be present in the sliding window as many times as the ones closer to the middle when using a filter matrix bigger than 1×1. This can be solved using zero-padding which is the process of creating an artificial border around the input image matrix filled with zeros so that the sliding window gives all the real input pixels the same amount of attention.

If we use a padding border with a width of 1 for the example with a 7×7 input, a 3×3 filter, and a stride of 1, we make that input into an 9×9 matrix (that is with padding) and we will get a 7×7 final output instead of a 5×5 one (resulting from the non-padded 7×7 input). Padding is also necessary for deeper neural networks because otherwise the input of deep layers would be too small.

ReLU

A ReLU function is used to introduce nonlinearity into the CNN. This is beneficial because the semantic information in the image is nonlinear too. The ReLU function is defined below.

$$\text{ReLU} = \max(0, x)$$

To explain how ReLU is useful in a CNN we use an example. We might be searching for a pattern of a dog in an image. When we examine one region, it

outputs a negative or zero value if the dog pattern is not found and positive if it is. The ReLU then makes it so that all the negative values are set to 0. Consequently, the output only activates if a pattern is found, which is great for finding complex patterns in an image. This is because finding only the head of a dog in one region is not as strong of an evidence as finding the head in one region and other parts of the dog in other regions and having higher confidence in the next layer that a dog exists in the image.

Pooling

The pooling layer is used in a CNN for the purpose of down-sampling the input and reducing its complexity for the following layers. One of the most common types of pooling is max-pooling which divides the input image matrix into rectangular regions of size N×N (most commonly 2×2) and selects the highest value among them. This highest value replaces the entire rectangular region in the output. If we take for example 2×2 pooling with stride 2 and a 4×4 input matrix we will get a 2×2 matrix as a result. This process is demonstrated in the figure 2.3 below.

Figure 2.3: input and output of a pooling layer

$$\begin{bmatrix} 1 & 1 & 2 & 4 \\ 5 & 6 & 7 & 8 \\ 3 & 2 & 1 & 0 \\ 1 & 2 & 3 & 4 \end{bmatrix} \longrightarrow \begin{bmatrix} 6 & 8 \\ 3 & 4 \end{bmatrix}$$

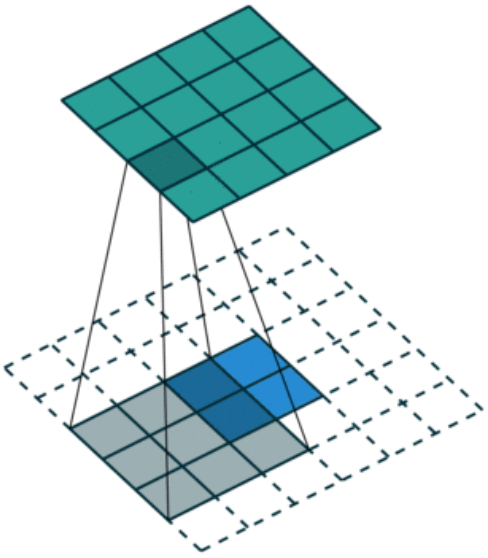
This example of pooling without overlap should be used only when the presence of information is what matters and the spatial information does not matter, because it does not retain that information (or rather it does retain it in a degraded/simplified form). Other types of pooling where the stride is smaller than the width of the rectangular region are used as well. Those can be more effective as they have some amount of overlap.

Deconvolution layer

The deconvolution layer or transposed convolution layer as it is sometimes called is an inverse layer to the convolution layer. It tries to predict the output matrix with knowledge from the input. As shown in the figure 2.4 it moves a sliding window matrix of dimensions bigger than 1×1 (as otherwise, the output would not be bigger than the input) across the input. Note that we first have to add padding with zeros to the input in order for this process to work.

Figure 2.4: deconvolution process visualized

(a) blue is the input and green is the output, white dashed line sqares area is the padding with zeros, image source: [18]



Unpool layer

Similarly to the deconvolutional layer, unpool layer works inversely to its counterpart – the pooling layer. For example in the figure 2.5 we have unpooling with a 2×2 kernel, stride 2 and a 2×2 input which results in a 4×4 matrix output. First, we create a matrix filled with zeros of the same dimensions as the future output matrix (4×4 in this case). After that, for each of the numbers in the unpooling input matrix we look at the position it was taken from as a maximum in the previous maxpooling layer and fill the corresponding position in our zero-filled matrix with the number from the unpooling input matrix. The new matrix filled with the numbers from the unpooling input is now the unpooling layer output matrix.

Figure 2.5: input and output of an unpooling layer

$$\begin{bmatrix} 9 & 7 \\ 7 & 8 \end{bmatrix} \longrightarrow \begin{bmatrix} 9 & 0 & 7 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 8 \\ 7 & 0 & 0 & 0 \end{bmatrix}$$

Fully-connected layer

The fully-connected layer which is also sometimes called dense or linear layer has its neurons arranged in a way so that each of the neurons creates a connection between every input neuron (in the previous layer) and every output neuron (in the next layer). It is typically used at the end of a deep CNN because it can create features with stronger capabilities from all the features previously extracted with convolution and pooling layers.

Layers in our CNN

In our non-hydra integrated CNN (the default pipeline, where the task is trained alone) we use a backbone with 8 convolution layers. Before each of the convolutional layers, a dropout of various magnitude is applied to reduce overfitting. After each convolution, a batch normalization over a 4D input (BatchNorm2d) from the PyTorch library is applied. Finally, the output is transformed with the ReLU function.

The convolution layers in our CNN are defined as Conv2d classes from the PyTorch library. The parameters of those layers along with the corresponding dropout are described in the table 2.1. Note that the IN denotes the input channels and the OUT the output learned channels of the layer. Note that before the output of the third convolutional layer is passed as input to the fourth layer, a coord convolution is applied. Coord convolution (or CoordConv) is an extension of the input via concatenation with two extra channels. Those channels contain coordinates in the form of hard-coded channels (numerical values going from 1 to 0 depending on their coordinates), where one goes horizontally and the other vertically. This adds spatial information which can help the CNN in conditioning based on location.

Table 2.1: parameters of the convolutional layers in our CNNs backbone

order of layer	**IN**	**OUT**	**kernel size**	**stride**	**padding**	**dropout**
1	3	32	3×3	2	1	0.1
2	32	32	3×3	2	1	0.2
3	32	48	3×3	2	1	0.3
4	48 + 2	48	3×3	1	1	0.4
5	48	48	3×3	1	1	0.5
6	48	64	3×3	1	1	0.4
7	64	128	3×3	1	0	0.3
8	128	128	2×2	1	0	0.2

The output of the backbone is the input to the last layer called the head (modified Conv1x1Head from PyTorch). The head is also a Conv2d class with the following parameters. A dropout of 0.4 (or a 40% probability of an

element to be zero-ed), 128 input channels, 1 output channel, 5×5 kernel size, 0 padding and a $(1, 1)$ stride. Note, that in the tuple stride, the first value translates to the height and the second to the width dimension.

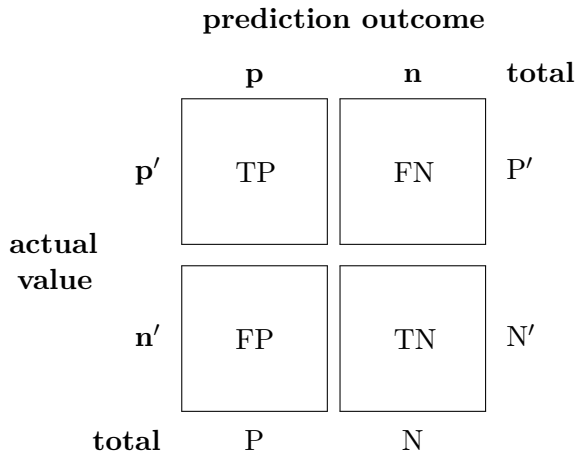
2.1.2 Other features

In this subsection we describe features of our CNN, which do not fall into the previous subsections, but still need to be discussed. Those features include dropout, classification accuracy measurement methods,

Evaluation methods

We use a variety of evaluation methods to measure the performance of our binary classification. Those can be explained more clearly with the use of the confusion matrix below in the figure 2.6, wherein our case p denotes the class pairs, n denotes the predicted class singles, p' denotes the ground truth pairs and n' denotes the ground truth singles. TP denotes True Positive, FN denotes False Negative, FP and TN denote False Positive and True Negative respectively.

Figure 2.6: confusion matrix



The most basic method is accuracy, which is simply the total number of correct predictions divided by the total number of predictions calculated as follows.

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{FP} + \text{FN} + \text{TP} + \text{TN}}$$

Next we used two relevancy based methods, recall and precision. Recall can be thought of as a measure of quantity while precision as a measure of quality. They are calculated as follows.

$$\text{recall} = \frac{\text{TP}}{\text{FN} + \text{TP}}$$
$$\text{precision} = \frac{\text{TP}}{\text{FP} + \text{TP}}$$

The last evaluation method we use is the F1 score. It is a measure of the model's accuracy calculated from its recall and precision. The lowest possible F1 score is a 0 and the highest possible is a 1. Note, that the F1 score has been criticized for giving equal importance to recall and precision, even though for some applications recall is more important and vice versa. For this reason, it is still important to examine all the measurements even when F1 score combines them. The F1 score is defined below.

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

Note, that there are other F scores that can be used when we prefer some metric over another such as the F2 score which weights the recall twice as important as precision.

Dropout

To reduce overfitting, our CNN uses dropout (specifically PyTorch Dropout2d class). Dropout is the practice of zeroing-out entire channels (in this case 2D feature maps) inspired by averaging models, which can help with generalization. Notice that in our CNN (table 2.1) we use very non-standard percentage values of dropout. In theory dropout of the hidden layers should progressively get higher the deeper the layer until it reaches a probability of retention of around 0.5 (or 50 %). After experimenting, we found that the best known dropout configuration for our network was sort of a hill-like process with increase from the dropout of 0.1 in the first layer reaching the maximum of 0.5 dropout in the fifth layer followed by a decrease until the dropout reached a probability of 0.2 in the eighth layer.

Note that the aforementioned dropout configuration is only used when our model is not trained within the hydra framework. This is because when integrated into the hydra framework, all the tasks (including our occlusion task) had to use the same backbone with a more standard dropout configuration of [0.0, 0.0, 0.1, 0.2, 0.2, 0.2, 0.2, 0.3, 0.3] (starting from dropout of the first layer and ending with dropout of the eighth layer).

Batch normalization

Another method that can help with overfitting, as well as vanishing gradient in a CNN, is batch normalization (commonly referred to as batch norm). Classical data normalization is a preprocessing technique that compensates for differences in the data by transforming them to use the same scale. Batch normalization is similar with the slight difference being that instead of normalizing the raw input data, it normalizes the contributions to layers in a

given mini-batch. This effectively speeds up the training because it reduces the internal covariate shift of the network so that fewer epochs are needed for convergence.

In our network, we use the BatchNorm2d batch normalization from PyTorch. It is applied each time after the convolutional (Conv2d) layer. Generally, batch normalization uses the following formula to normalize each of the layer's outputs in a given batch.

$$x_n = \frac{x - E[x]}{\sqrt{Var[x]}} \cdot \gamma + \beta$$

Where x_n is the normalized output, E[x] is the mean of the neurons output, $\sqrt{Var[x]}$ is the standard deviation of the neurons output, x is the input to the barch norm and finally γ and β are the learnable parameter vectors (with sizes of C, which equals to the size of the input).

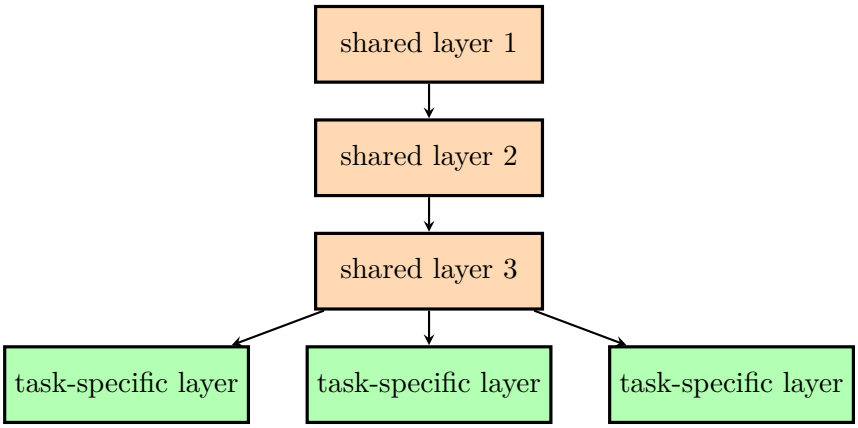
2.2 Hydra architecture – multi-task learning

The general approach to machine learning problems is to train a model to perform a single task, measure some score and tweak the model to achieve the highest possible performance. Sebastian Ruder in his work An Overview of Multi-Task Learning in Deep Neural Networks[19] describes that a better approach might be to share representations between tasks which makes the model generalize better on the desired task because in this way we do not ignore some of the beneficial information and the model has to leverage domain-specific information between related tasks (for example between a task of finding the faces of pedestrians and a task of finding two occluded pedestrians). Multi-task learning has been successfully applied to fields such as NLP, speech recognition, and most importantly for us, camera vision. Generally, whenever we are trying to optimize multiple loss functions, we are doing multi-task learning. Thus many different approaches can be called multi-task learning such as joint learning or learning to learn.

2.2.1 Motivation

To try to understand why it works, we will first look at multi-task learning from an intuitive perspective. When humans learn, they often use previously learned related knowledge to ease their learning. For example, a baby learns to recognize faces and then uses this knowledge to learn to recognize different objects. This is also true for adults, let us use the film *The Karate Kid* as an example. In the movie, the old sensei teaches his apprentice many tasks seemingly unrelated to karate which secretly build the general foundation skills (such as coordination and perseverance) of the apprentice so that he can then perform karate much better.

Figure 2.7: multi-task learning via the hard sharing method



From a machine learning perspective, we have an incentive to make the model generalize better. Multi-task learning helps with that by introducing inductive bias which favors some hypotheses over others. In the case of multi-task learning, it favors those that can explain more than one task, which leads to models that generalize better.

2.2.2 Multi-task learning methods

The two multi-task learning methods for deep learning are differentiated based on their approach to parameter sharing. The first method uses hard and the second soft sharing of parameters of the hidden layers.

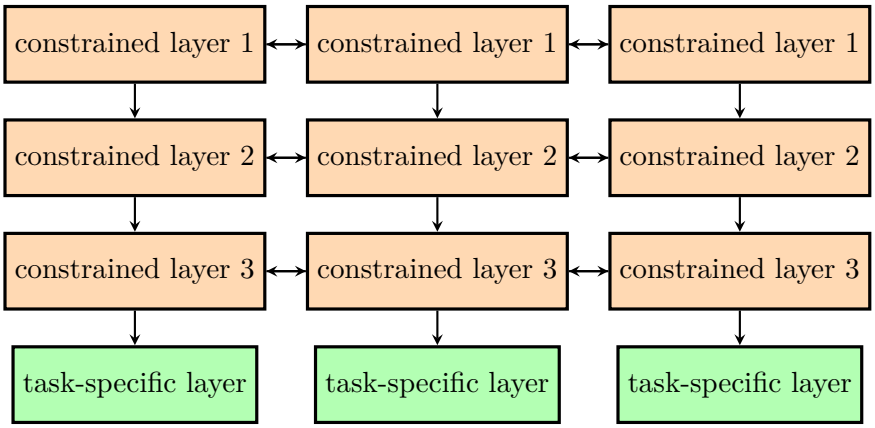
Hard parameter sharing

As described in the layer schematic2.7, hard sharing uses the same hidden layers for all tasks with task-specific output layers. It is the most common approach and greatly reduces overfitting because the model has to find representations that fit all the tasks.

Soft parameter sharing

In soft parameter sharing, there exists a separate model for each task, but each model is encouraged to have parameters that are similar to the parameters of other models. This encouragement is achieved with the use of regularization of the distance between the models. For example, the L^2 distance or the trace norm can be used.

Figure 2.8: multi-task learning via the soft sharing method



2.2.3 Multi-task learning advantages

Ruder in his work[19] lists several advantages that multi-task learning brings to neural networks and describes their underlying mechanisms. For the following examples, consider two related tasks A and B which both rely on a common representation R from a hidden layer.

First let's examine the implicit data augmentation, which means that we will effectively have a bigger sample size. This is because both task A and task B have some noise patterns and when trained simultaneously the model will learn a more generalized representation F through noise pattern averaging.

Another advantage of multi-task learning is that it focuses attention on the features that are really important because all the tasks provide evidence to each other about the relevancy of the various features. This can be advantageous when we deal with tasks that have complex and high-dimensional data.

Some features (denoted as G) might be easier to learn for task B than they are for task A. This is because other features of A might be discouraging the task from learning G or because task A is interacting with the features in a different way than B does. The advantage of task A learning a feature G through task B learning it is referred to as eavesdropping.

Finally, multi-task learning introduces representation and inductive biases. This means that the model will prefer representations that fit all the tasks, which helps generalization on new tasks as the model that already fits many tasks will perform better on new tasks as well. The inductive bias causes regularization which reduces the risk of overfitting.

2.2.4 Our hydra architecture

We call our multi-task learning system hydra after the many-headed beast from Greek mythology because just like that monster our architecture has one backbone shared between the tasks and a multitude of heads, one for each task. The multiple purpose heads add output features to the model. Those features can be later used to optimize certain tasks. Our pairs classifier head will output a number between 0 and 1 which will tell us the confidence of the proposed region being a pair or a single. This additional information can be useful for example in the task counting, which consists of detecting and tracking pedestrians and then counting how many of them have gone in and how many have gone out of the shopping center. In the counting task, the domain is separated into counting regions, where the total count is updated. When an occluded pair is present in the counting region, the output of the pairs classifier head can help decide if the proposed region contains a pair or a single pedestrian, in effect making the counting task more accurate. Other heads, which are the work of my colleagues are predicting the gender of the pedestrians or their age and other information useful in various tasks. Note, that due to time constraints, all the experiments run within the hydra framework, described in this thesis, are trained with just the pairs task purpose head.

Chapter 3

Datasets

In this chapter, we first explain our approach to dataset creation, then describe our heuristics, which we use to create our datasets or to enlarge our datasets. In the section Datasets, we explain the differences between various datasets. Finally, we explain the problem of the imbalance of classes present in our datasets. Note, that this chapter is closely tied to the chapter 4, specifically to its section 4.1, because its experiments are done during the dataset creation phase.

3.1 Problem definition

Before we could even begin to start solving our problem and improving our detector, we had to properly define what we are trying to solve. We wanted to improve an existing detector of pedestrians and our improvement would have to work within the confinements of the hydra framework. Our solution is to use the existing detector to predict regions which are then classified as having either a pair or a single person in them and this information is passed on in the form of a confidence score. Based on the confidence score the final detection is then doubled (with some variation/jiggle to prevent unstability) if our classifier returns high enough confidence of a pair being present.

The problems with this definition lie in the dataset creation phase. This is because we create all our datasets from the BBox annotations. Those are human annotations drawn on data from iC systems.ai cameras in the form of boxes around each detected pedestrian. We explain how exactly we gain the data from those annotations in more detail later but in essence, we measure the overlap of two boxes in an image and decide if it should be added to our dataset as an occluded pair sample or as a single entirely based on this overlap.

This is a problem because the classes are not discretely separable. For example, with an overlap threshold of 9% one sample of a pair could have an overlap of 50% and another sample of a pair might have an overlap of 10%, the difference being 40% of an overlap. A sample of a single might have an

overlap of 8%, which is a difference of just 2% from the second pair sample. Of course, this difference by itself might not cause any problems, it just illustrates how much disparity there is between samples within the same class. We tried to make the problem more discrete by setting a margin between the overlap of pairs and singles samples. The singles have to have a maximum of 0% overlap and the minimum overlap threshold (which equals a margin between a pair and a single) for the sample to be considered a pair changes across datasets (this is described later in the section 3.3). This is because, with a bigger overlap margin, the classes are more clearly defined but the dataset size of pair samples shrinks and the count of pairs is already disproportionately smaller than that of the singles samples so we wanted to try more versions of margins to see which dataset performs the best.

In this chapter we first explain the heuristics used to create our datasets, then we explain the problems with our unbalanced datasets and how we tried to solve them with cleaning and augmentations. Finally, we conduct various experiments and discuss their results.

3.2 Heuristics for dataset creation

3.2.1 Picked detections heuristics

For the purpose of training, iC systems.ai uses its own data annotated in different ways on a remote server with the help of human annotators. The annotation task performed to gain the data for the purpose of this paper is called BBox+ and requires the annotators to draw a box around the pedestrians present in the image. As the process of annotation is costly, we tried to gain as much useful data as possible. Making use of the automatic tracking data, the following heuristics were used to pick our detections that were suspicious of having occluded pairs of pedestrians.

In order to understand the heuristics below, it is first necessary to understand the data structure of tracking. Each track represents the path of one object which can exist across multiple frames. This path is represented with detections. Detection mainly represents the position of the detected object in the frame but also has other parameters. These include the frame in which the detection lies, ground point coordinates, and region of interest. The region of interest contains the X and Y coordinates for a rectangle representing the object's silhouette. Note that the path doesn't have to be continuous, the detector can recognize a detection that belongs to the same track after it has lost it for some frames. The images forming videos that correspond to frames in tracking are kept separately from tracking data.

The first version of the heuristic[2] tries to find pairs of pedestrians traveling in the same direction, where one of them is missing for some frames, and picks

[2]both versions can be found inside the file *pairs_heuristic_OUTPUT_is_frames+dataset.py*

out those frames. We go through all the detections gained with automatic tracking and transform them into a format containing suspicious detections and directions for each frame. Directions are calculated using the difference between the last and the first X and Y coordinates (denoted as last_x and first_x for the X coordinates) of the track. After that, we get the arc tangent of those differences which is then converted from radians to degrees as can be seen below.

$$\begin{gathered}\text{direction} = \text{degrees}(\text{atan2}(\text{diff}_x,\text{diff}_y)), \text{ where} \\ \text{diff}_x = \text{last}_x - \text{first}_x \text{ and} \\ \text{diff}_y = \text{last}_y - \text{first}_y\end{gathered}$$

This is done because we can set a threshold for the difference in the direction of two detections later on. In the next part, we check each frame of each track for its detections and pick out detections from the suspicious frames. Suspicious frames are the ones that are missing in the tracking, which means that the object of the tracking was followed then lost, and then picked up again. This makes it more likely that the object was occluded. We also save an array with the directions of all detected detections (this combination of words might seem obsolete, but it is very important – it indicates that the undetected detections, which may still be present, are not in this array) in each frame.

Now we have set up our structures, an array with last seen detections of suspicious tracks and their likely directions indexed with frames and another array with directions of detected detections indexed with frames as well. For each suspicious frame, we now examine the direction of its supposed missing detection and the direction of its potential occluding partners. If the directions fall within the threshold cone of 70 degrees, the frame is finally flagged as suspicious and the corresponding image is uploaded to the server to be annotated.

The second version of the heuristic picks out all frames where two pedestrians are close enough. Again, we go through all the tracks gained with automatic tracking and transform them into detections indexed by frames. With this new transformed format, we go through all the frames and calculate the Euclidean distance of their ground points. Ground points are estimated X and Y coordinates of the pedestrian's contact with the ground and are already present in the automatic tracking. We then compare the smallest distance between any two detections in a frame to a preset threshold. If the smallest distance falls below, we upload the corresponding image to the annotation server.

3.2.2 Heuristic to create datasets from annotations

This heuristic[3] is used after annotators assign boxes to all objects in all sent frames and we have to decide which ones have examples of tuples or singles and transform them into the final format of an array of crops. We refer to the boxes created by annotators as detections below.

We go through all the frames and examine each detection (denoted A) in relation to other detections (denoted B) on that frame. First, we find the maximum percentage overlap of A and B described below.

$$\text{percentage overlap} = \frac{(O_x * O_y * 100)}{\theta_A}$$
$$\theta_A = x_{abs} * y_{abs}, \text{ where}$$
$$O_x = \max(0, \min(\delta_{x,\,A}, \delta_{x,\,B}) - \max(\gamma_{x,A}, \gamma_{x,B}))$$
$$x_{abs} = \text{abs}(\gamma_{x,A} - \delta_{x,\,A})$$
$$\text{and likewise for y (instead of x).}$$

Where $\gamma_{x,A}$ denotes the X coordinate of the top left corner of the detection A and $\delta_{x,\,A}$ the X coordinate of the bottom right corner. θ_A denotes the full area of A and κ the overlap area. O_x denotes the x overlap. For the detection B and coordinates Y, the denotations are similar (with the change of A to B and X to Y).

Next, we find the corner of the detection that is closest to the center of the image, because the cameras point downwards so whatever object is closest to the center should be occluding others. Afterward, we compare it to the corners that are closest to the center of the image of other detections. This way we are able to gain information about the occlusion category of that detection. When the distance of A is closer to the center than that of B, A is occluding B. If the opposite is true, A is being occluded by B. So in the end all detections in a frame have two boolean variables assigned to them, is_occluded and is_occluding. We may use this for some heuristic or as a parameter for our network later on.

Finally, we use the previously gained information about the maximum percentage overlap to determine if a detection is a sample of a pair or a sample of a single. Detections that are determined as being pairs are detections where there are two or more people occluding each other and are always denoted as zeros in our datasets and experiments. Detections determined as being singles are detections where there is only one pedestrian with no amount of occlusion. Whether a detection is a pair or a single is decided using a threshold of the percentage of occlusion. Detections with a maximum overlap percentage higher than that threshold are determined as pairs and detections with a maximum overlap percentage equal to zero are determined as singles. After the decision whether a detection is a pair or a single is made, we cut out crops from the full image with a small margin around the detection and save them.

[3] can be found inside the file clf_reg_gt_generator_NEW.py

This is the final step in the creation of our datasets. This creates a margin between pairs and singles so that the decision isn't continuous (meaning that the difference between a pair and a single could theoretically be infinitely small) and it is easier to train our network.

The threshold and subsequently the margin of percentage between pairs and singles greatly affects the created datasets. When it is too low there is less difference between a pair and a single. But as we are working with human-annotated data, it cannot be too high either, because we have little data available. Because of this, we create different datasets based on the margin between the singles and tuples and as this name is too long, we will further refer to a dataset created with this heuristic with a margin between pairs and singles of $XX\%$ as "with margin 0.XX".

3.3 Datasets

In this section, we describe how the various datasets used in our experiments were created. We further describe how well they performed in general and try to find an intuitive explanation as to why that was the case.

iC systems.ai has cameras installed in several shopping centers. Those cameras and their hardware perform detections of pedestrians and this is where we get our training data from. The cameras are installed in such a way that they look directly down, meaning that the center of the image is also the closest ground point to the camera (provided there is a flat surface).

The crops of pedestrians from different quadrants of the image are mirror transformed so that all of the pedestrians have feet and heads in roughly the same direction. This transformation is done because we want the pedestrian crops to be roughly aligned. Another challenging example present in our datasets is the top-down pedestrian (example in figure 3.1) which the camera captures walking directly below itself. This can be problematic because the shape of the pedestrian is different from all the other angles and less of the pedestrian's body is visible so there is less information. On the other hand, those types of positions should have little to no occlusion.

Figure 3.1: example of the top-down pedestrian type

3.3.1 Datasets created from annotated picked detections

The heuristic above was also used to create datasets from picked detections, as those were smaller in count, we didn't attempt to try them out on their own. Instead, they were joined with corresponding datasets (with the same margin between singles and tuples). As was the case above, we want to simplify and codename this dataset, so we will further refer to a joined dataset of picked and other detections with a margin of $XX\%$ as "with margin 0.XX with picked detections".

The addition (via simple concatenation) of the picked detections to an already existing dataset led to an increase in accuracy, which is unsurprisingly caused by the small training dataset of zero samples (pairs) increasing in size.

3.3.2 Validation and train datasets created from separate camera sources

The first dataset that we experimented with was created in the following manner. First, we separated all of our available image data according to the shopping centres in which they were collected. Then we picked one of those shopping centres, where the count of images was roughly in accordance with the usual ratio of a validation dataset in proportion to the training dataset. Then the validation dataset was created only with the data from this selected shopping centre and the training dataset only with the data from the rest of the shopping centres. To simplify, we will refer to a dataset created in this way with the suffix "_val=separate"

These datasets were not as successful as the ones described below so we stopped using them during experimentation. They would have uses if we were trying to build a network for the general use case of recognizing occluded people, but that was not our case. We were mainly trying to build a system that performed as well as it could have in the few shopping centres that ordered our services. This type of dataset creation may have its uses in the future.

Table 3.1: dataset sizes of the dataset with separate validation with margin 0.10

dataset	**# of single samples**	**# of pair samples**
Train	26240	1404
Val	3948	350

Note, that the addition of picked detections to this dataset increased the size of Train-pair samples with 1094 samples from 1404 to 2498.

3.3.3 Validation and Train datasets created from the same camera sources

Unlike the previous datasets, these datasets were created by random sampling from the image data from all the shopping centres. To avoid confusion, even though we tried them out as second, this process of creation can be thought of as the standard in classification problem dataset creation. They immediately increased performance compared to the datasets with separated data for validation and training. This is not surprising as the camera angles in different shopping centres are varied so complete generalization of the task is not possible. This is the dataset that we used the most in our experiments.

Table 3.2: dataset sizes of the dataset created from the same camera sources with margin 0.10 with picked detections

dataset	**# of single samples**	**# of pair samples**
Train	19813	2388
Val	4984	627

3.4 Unbalanced data

Both our training and validation datasets were sampler-level balanced, meaning that each batch consisted of the same amount of pairs samples as the amount of singles. This balance was forced unnaturally because our dataset contained many more singles than pairs, which would result in the model not properly learning from the pairs samples. However, in reality, the data are not balanced and thus the performance of our model on a balanced validation dataset is misleading because it does not reflect the performance if the model would be used in real-world conditions.

To better reflect reality, we applied imbalanced (unchanged and natural) sampling. The amount of samples of pairs and singles in a given batch was now sampled in proportion to the real dataset sizes, which were massively imbalanced. To force the model to learn to classify pairs we used weights to amplify their impact in the loss function in the following way.

$$W_1 = \frac{1-|\{y=1\}|}{|\{y=1\}|+|\{y=0\}|}$$
$$W_0 = 1 - W_1$$

Where W_1 denotes the weight with which we multiply the loss of the samples of singles and W_0 denotes the weight with which we multiply the loss of the samples of pairs and $|\{y = 1\}|$ denotes the count of samples with ground truth set as a single and for pairs likewise.

Chapter 4

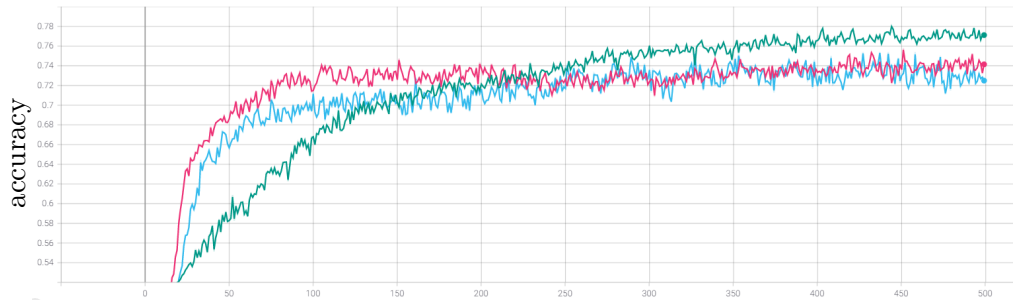
Experiments

4.1 Datasets experiments

The experiments in this section are done during the dataset creation process described in the previous chapter and as such, they explain the differences between different versions of the datasets.

4.1.1 Accuracy comparison across different margins between singles and pairs measured on the dataset created from the same camera sources

Figure 4.1: TensorBoard visualization of comparison of validation accuracy between datasets with 0.2 and with 0.1 margin between singles and pairs over training epochs



performance of datasets with margin 0.10 with picked detections (green), with margin 0.20 with picked detections (light blue) and with margin 0.30 with picked detections (pink)

As we can see in the figure 4.1 The smallest margin between a ”pair” and a ”single” sample had the best performance overall (achieving the best accuracy

score of 78.5 %). However, this can be deceiving as this is mainly caused by the small dataset size or rather by the 0.10 margin dataset being larger than the two much smaller ones. The 0.30 margin dataset trained the fastest as it had the least samples but stopped improving relatively quickly. This was also the case with the 0.20 margin dataset to a lesser degree.

Table 4.1: train dataset sizes of datasets with different margins

margin	**# of pairs in train dataset**	**# of pairs in validation**
0.10	2388	627
0.20	1264	344
0.30	685	192

Note that the number of singles is not needed in the table 4.1 as it remains the same, margins changes only how many samples of pairs we get.

4.1.2 First results on a dataset with unbalanced validation sampling

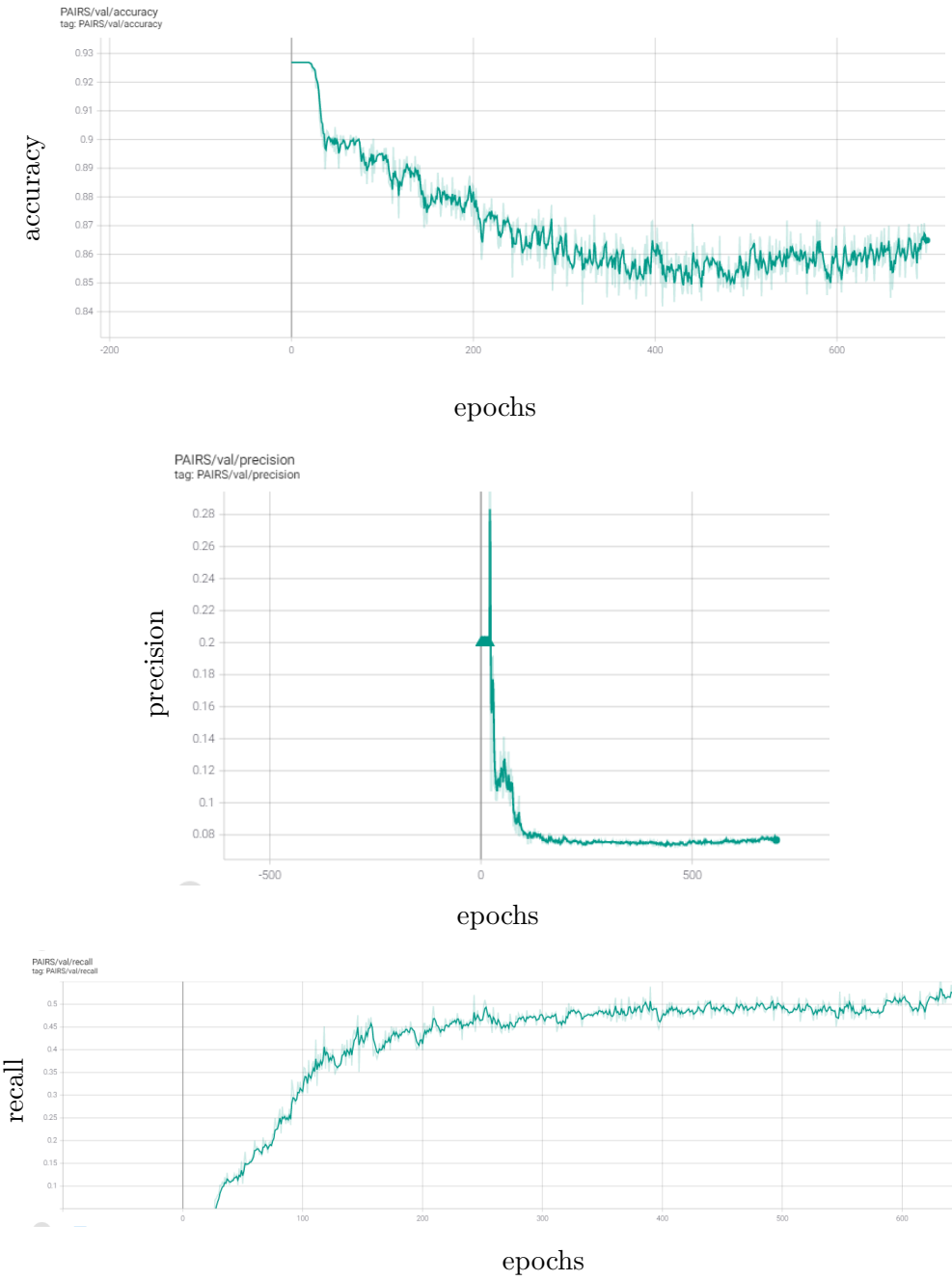
Because we previously achieved meaningful scores of around 70% precision, 85% recall, and 75% accuracy after 500 epochs we were at first surprised to find that with the new imbalanced sampling we were no longer anywhere near those results. As we can see in the figure 4.2, which visualizes the values of accuracy, recall, and precision on the dataset with margin 0.20 with picked detections the precision as well as the recall is really bad.

Looking at possible solutions to those bad results, we first tried other datasets but those bad results were repeated with all of our datasets (with different margins of overlap between a single and a pair). As the imbalanced validation dataset was a correct approach and thus we didn't want to change it back, we looked for other reasons that could cause the model to have such a low precision and recall. Finally, we visualized the data and found out that a lot of the samples were mislabeled which will be discussed in the next section.

4.2 Ignoring bad samples

As mentioned in the previous section, after we visualized the data samples that had the biggest loss, we discovered that a large part of our datasets is mislabeled or labeled correctly but has a lot of noise to the point of being detrimental to the training. To increase our performance as well as make the validation measurements more accurate we manually went through the images and picked out their IDs. The picked out image IDs are then inserted into a separate file and skipped during the loading of the datasets and in this way the bad samples are ignored. Note, that due to our method of dataset

Figure 4.2: TensorBoard visualization of various validation metrics (accuracy, precision, recall) when using weighted imbalanced-sampled dataset for training and validation

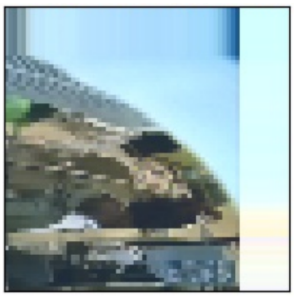


creation, the IDs are different across different datasets so we decided to pick out the wrong IDs and subsequently clean only one dataset, specifically the 0.1 margin overlap dataset as this dataset performed the most consistently during our previous experiments and had the most samples.

cleaning the pairs samples

As the pairs samples are smaller in number than the samples of singles, we had to be more careful with picking out the wrong IDs, because we wanted to pick out the least amount possible. For this reason, we manually classified (based only on the judgement of a human) the samples into four categories – right, wrong, weird, and separate. The right ones are the samples that are the samples that we want and are not ignored in any of our experiments. The wrong IDs are of the samples that are obviously wrong. In most instances, this means that a clear single pedestrian is present in the crop instead of an occluded pair or that no one is present in the crop at all. The wrong IDs are ignored in all of our following experiments. The weird samples IDs are picked out for a multitude of reasons, but most of them have a huge amount of artifacts (usually because the annotators annotated pedestrians on the edge of the image and the camera lens caused distortion like in figure 4.3) from the camera and can't be considered proper right samples even though occlusion of pairs might be present. The IDs that are classified as separate have no occlusion present in their corresponding samples and their supposed paired pedestrians have a clear distinction margin between their bodies. We experimented with the skipping of the last two types of samples but ultimately found out that skipping them both is beneficial to the training. This picking strategy concerns the training and validation datasets containing samples of pairs internally referred to as train_0 and val_0.

Figure 4.3: example of the artifact/distortion type of bad sample



cleaning the singles samples

Note, that this concerns the training and validation datasets containing samples of singles internally referred to as train_1 and val_1. Because the datasets

containing singles are much bigger, we chose a different picking strategy than with the pairs samples. Instead of picking out the wrong IDs, we pick out the right IDs, which means that we manually go through the samples and select only the examples which contain single pedestrians with no occlusion and little to no image artifacts. An example of a mislabeled single sample can be seen in the figure 4.4, where the sample clearly contains a pair with a significant amount of occlusion but it is labeled as a single.

Figure 4.4: visualisation of a misslabeled sample of a class single. Values from left to right: error, predicted value, ground truth, id of the sample.

Results of the cleaning

The cleaning of the datasets had a huge impact on the performance as can be seen from the figure 4.5, where a comparison of the evaluation metrics precision, recall, and accuracy in various stages of dataset cleaning can be seen. First, we cleaned the full pairs train dataset (train0) and a 100 samples from the pairs validation dataset (val0) with the biggest error after one epoch – this is the orange plot in the figure. Then we additionally cleaned the singles train dataset (train1) which is visualized as the pink plot. Finally, we cleaned the singles validation dataset (val1) and the rest of the pairs validation dataset (val0) – this is visualized as the grey plot. The datasets are again summarized below for ease of reading because the datasets are referenced several times in the following text.

- orange – cleaned train0 and a 100 samples with the worst error from val0
- pink – cleaned everything above and additionaly train1
- grey – cleaned everything above and additionaly val1 and val0

Figure 4.5: TensorBoard visualisation of the results from datasets cleaned to various degrees

(a) orange=least cleaned, pink=more cleaned, grey=most cleaned – the datasets are described in more detail below

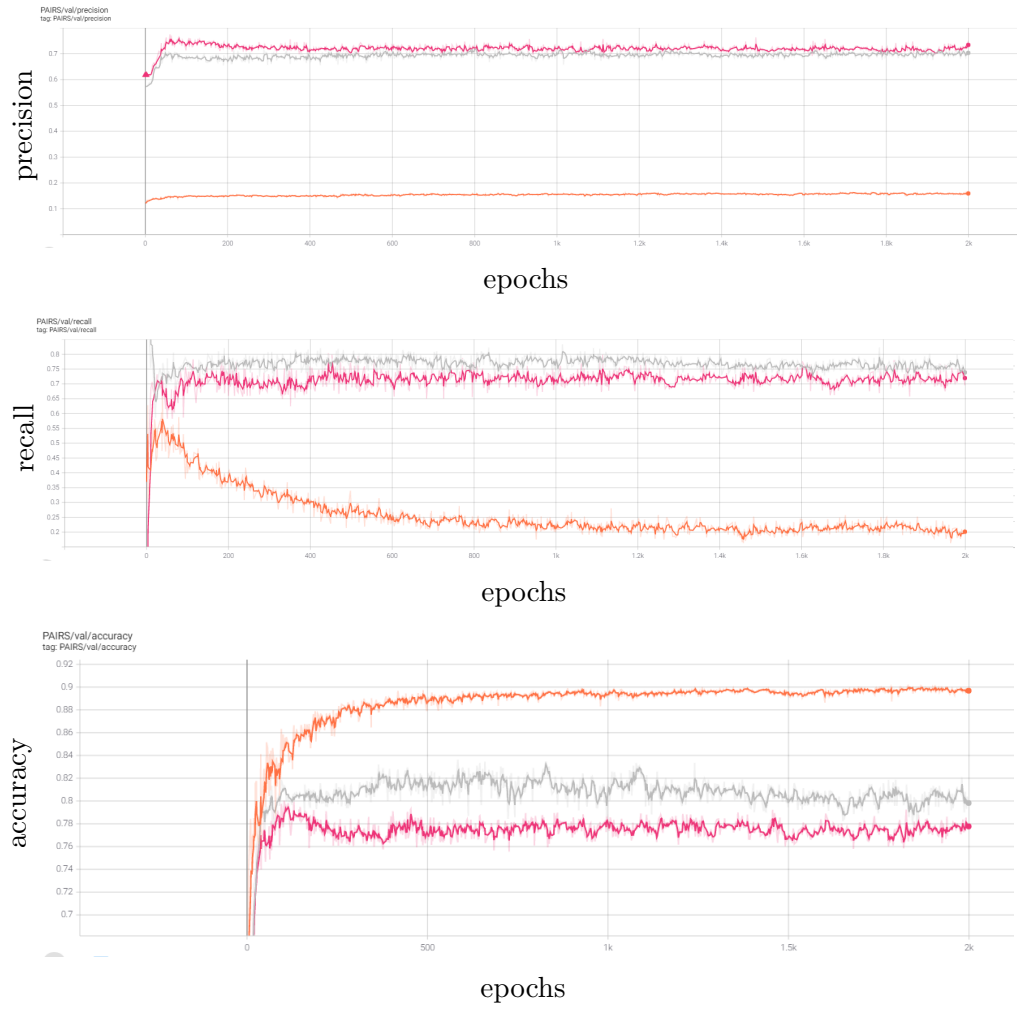


Table 4.2: rounded ratios of pairs to samples in a validation batch in different stages of cleaning

stage	**rounded ratio of pairs to singles**
1. – orange	1 : 15,3
2. – pink	1 : 1,47
3. – grey	1 : 2,22

To better understand the validation results from the various datasets, we need to take into account the class ratios, since those have a significant impact on all the metrics. The ratios are listed in the table 4.2. The first version of the cleaned dataset (orange) has a really imbalanced ratio in validation. The imbalance in favor of the singles samples negatively affects the precision, because more of the misclassified pairs (false positives, positive being the class pair) samples end up being counted as wrong (specifically false positives), whereas if we had a more balanced dataset some of those misclassified samples would still end up being counted as correct (specifically true positives) simply because there is a higher chance of being a pair for each sample. Note that the real performance is still bad in the second case described above as the model doesn't classify any better than the previous one, but the evaluation metrics can differ greatly with various ratios so the ratios have to be considered when evaluating each of the datasets. With just the pairs samples cleaned (orange), we achieved 2% higher precision of 15% (2% increase in precision being an increase of 15% from the previously achieved 13% precision). This motivated us to continue with the cleaning process as precision was the most important metric for our purpose of improving the pedestrian detector.

The pink dataset with all the train samples (pairs and singles) cleaned achieves a huge increase in both precision and recall with a slight decrease in accuracy. This would mean a perfect result of our cleaning, but a huge part of the result has to be attributed to the change in class ratios, which are now 1 : 1,47 (pairs : singles) for the reason described in the paragraph above. Still, the results have improved, the lower accuracy can be explained with the changed ratio as well because the model now has a harder time predicting since before all the model had to do to achieve high accuracy was to learn to classify as many samples as singles.

The recall is much bigger after the cleaning of the training dataset as well. This metric is interesting as it shouldn't be higher simply because of the change in ratios since we use weighted loss, which means that the higher representation of singles samples in the training dataset is appropriately compensated in the loss function so that the models learn both classes equally. This means that the data cleaning really did improve the recall of the model to a high enough value usable in a real application.

In the grey dataset, validation has been fully cleaned which has the expected effect of an increase in recall and accuracy. Interestingly there is a small decrease in precision, but the decrease is small and it can be attributed to fluctuations in training.

Another advantage over the uncleaned models is that previously the model was never 'sure' about the predictions meaning that the predicted value was often close to an uncertain 0.5 (where 1.0 means a sure prediction of a single single and 0.0 means a sure pair) even in the later epochs. With the cleaned datasets, the model started to be differently sure about its predictions, meaning it has – at least in theory – really learned. This is useful for our purposes

because we want to improve the detector so the 'sureness' information can be used for setting of an additional threshold for classifying. For example, pairs can only be the detections with pairs confidence score above 0.9 (equal to a prediction score below 0.1).

4.3 Augmentations

To reduce overfitting as well as inflate our dataset size we use a variety of augmentations from the imgaug library. Augmentations are transforms of the sample images so that one sample can be used multiple times increasing the effective sample count. The transforms can range from changing the colors or shifting the image to adding various noises or artifacts to the image. It is important to note that in the case of our problem, some augmentations may actually be detrimental to the learning. This is due to the fact that in many of our image samples the interesting object (pedestrian or pedestrians) is small and for example, adding artifacts or shifting the image may entirely or partly remove the pedestrian or parts of the occluded pedestrian from the sample. The augmented sample of a pair would then become a sample of a single (or of an undefined class) but would still be labeled as a pair, confusing the learning process.

For this reason, we only use the augmentations that do not completely remove or occlude parts of the image. During training 50% of the images in a batch have an equal probability ($\frac{1}{3}$) of being transformed with one of the following augmentations.

- Adding a value in the range $\langle -50, 50 \rangle$ to all pixels in the image.
- Changing the contrast of the image with a strength value in the range $\langle 0.75, 1.75 \rangle$, where a value lower than 1 decreases the contrast and a value greater than 1 increases the contrast.
- Changing the color temperature of the image to a value ranging from 2500 (warmer, more red and yellow) to 8000 (colder, more blue). Note that only 20% of the one third of the 50% (so in total $0.5 \times \frac{1}{3} \times 0.2 = \frac{1}{30}$ or around 3%) of the images in a batch will be transformed in this way.

4.4 Hyperparameters

In this section, we experiment with different hyperparameters. First and foremost, we try to find the best model. Because of this, once we found out that some hyperparameter or dataset configuration was the best we usually stopped testing others of the same type and focused on changing different aspects. So as we gradually progressed towards better accuracy it is possible that some aspect that would actually yield better results in a different configuration has

been forgotten and has not been tried in the end, because the number of possible configurations and combinations of datasets is just too high and we mainly wanted to make the best model possible. Note that this approach is true for all our datasets experiments as well.

4.4.1 Testing various hyperparameters on the old dataset with separate validation data sources

We started our experiments with various hyperparameters on the old dataset with separate validation data sources described in the subsection 3.3.1.

Table 4.3: accuracy measurements with various hyperparameters on the separate validation dataset

change (compared to baseline)	**best accuracy**
baseline (100 batch size and adamW(lr=1e-3))	0.628
new aug (new augmentations)	0.607
new aug with 100 batch size	0.611
new aug with 64 batch size	0.618
new aug with 64 batch size and adamW(lr=1e-4)	0.622
baseline with 64 batch size and adamW(lr=1e-3)	0.604
baseline with 64 batch size and adamW(lr=1e-4)	0.622

First, we should explain the changes in hyperparameters. Previously we used a set of augmentations described in the section 4.3, the new augmentations are the following. The color jitter which changes the brightness, saturation, and contrast of the image. The coarse dropout which randomly drops small rectangular regions from the image. The fancy PCA which performs principal component analysis and alters the intensities of the RGB channels. The gauss noise which adds gaussian noise to the image and finally the random shadow augmentation which adds shadows to generalize better in various lightning conditions. We also experimented with different learning rates on the AdamW optimizer and with smaller and bigger batch sizes.

As we can see from the table 4.3, none of the changed hyperparameters reliably improves the accuracy. Our measurements show that the accuracy is worse than baseline with all the tried changes. That can be deceiving as the baseline accuracy score is the best out of several validation runs, whereas the scores of the models with changed hyperparameters were only evaluated two times, which means that the better score might be caused by fluctuations in learning. Still, as we did not see any improvements to the baseline accuracy, we used the baseline hyperparameter values.

After experimenting with this dataset, we later discovered that it did not fit our purposes of the final model performing on the select cameras as best

as it can as the model trained with this dataset would only perform better when applied to an unseen camera source (different, unseen shopping centre). This is why this dataset and its results are never used, experimented on, or discussed later.

4.4.2 Testing various hyperparameters on the current cleaned dataset with margin 0.10 with picked detections

As accuracy does not explain the model entirely we provide other metrics: precision, recall, and the F1 score as can be seen in the table 4.4. The model with the best achieved accuracy is the model, where the batch size is 100 and the learning rate is $1e^{-3}$. The model that achieved the best precision differs from the one with the best accuracy only by using the new augmentations during training. The model with the best recall differs from the one with the best accuracy by changing the batch size to 64. Note that all the models oscillated greatly and their scores were very similar for most of the training, the best achieved scores are mostly outliers often achieved during the first 100 epochs (out of 2000 epochs). Furthermore, the recall and precision scores are mostly negatively correlated, meaning that the lower precision means higher recall and vice versa. This means that the F1 score metric is more accurate and more telling.

The highest F1 score is achieved with the same model as the one with the highest accuracy. So the best configuration found so far is 100 batch size, a learning rate of $1e^{-3}$ and no attention in backbone or new augmentations used. The significance of this finding is not that important as the change in the F1 score is not large enough (so it could be caused by randomness and fluctuations in training). The smoothed progression of the F1 score of models with different hyperparameters across epochs can be seen in the figure 4.6. In the back of the figure and less visibly, the real values and their heavy oscillations can be seen.

4.5 Final tracking evaluation

Our ultimate goal is to improve the performance of the existing tracking algorithm with added information about occlusion. We measure the impact of our model described in the sections above by applying it to tracking in the counting task. In the counting task, the detector detects and tracks pedestrians, each pedestrian that goes inside some predefined zone from one direction is counted. The counting task was really useful during the pandemic when the

[1] which AdamW learning rate is used

[2] new augmentations used (same as in the previous experiment in subsection 4.4.1)

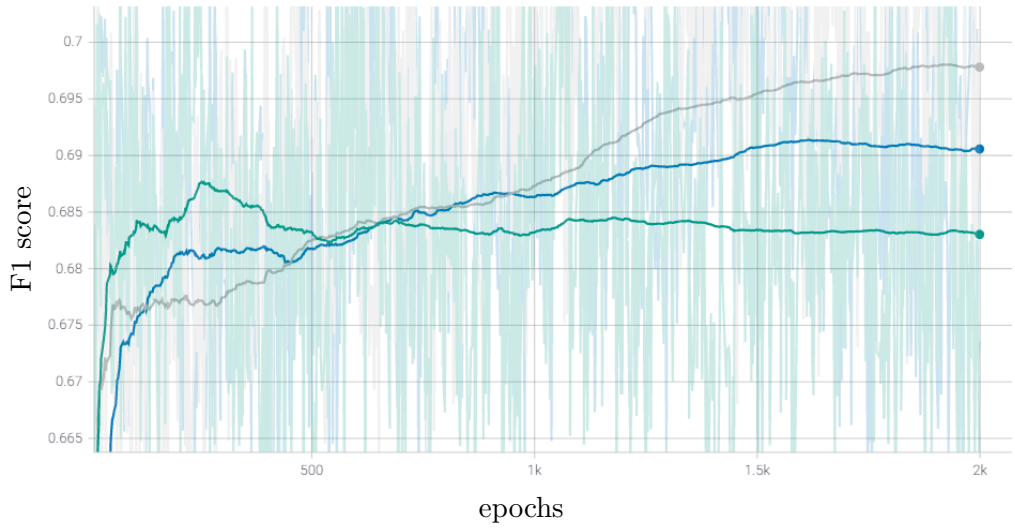
[3] backbone with added attention feature used

Table 4.4: accuracy, precision, recall and F1 score measurements with various hyperparameters on the current clean dataset

hyperparameters				validation results				ID
batch size	**learn rate** [1]	**new aug** [2]	**attention bb** [3]	**best accuracy**	**best precision**	**best recall**	**best F1-score**	
100	$1e^{-3}$	True	False	0.789	0.771	0.781	0.725	0
100	$1e^{-4}$	True	False	0.826	0.732	0.854	0.722	1
64	$1e^{-4}$	True	False	0.823	0.763	0.797	0.713	2
64	$1e^{-4}$	False	False	0.818	0.716	0.821	0.721	3
64	$1e^{-3}$	False	False	0.820	0.742	0.884	0.728	4
100	$1e^{-3}$	False	False	0.841	0.766	0.876	0.737	5
256	$1e^{-3}$	False	False	0.816	0.756	0.871	0.716	6
256	$1e^{-3}$	False	True	0.839	0.763	0.849	0.734	7
100	$1e^{-4}$	False	True	0.820	0.736	0.836	0.717	8

Figure 4.6: heavily smoothed (with smoothing factor of 0.999) TensorBoard visualisation of the three runs (with different hyperparameters) with the highest maximum F1-score

(a) grey run ID = 5, blue run ID = 7, green run ID = 4



number of people inside a shopping centre should not reach above a certain threshold.

The improvement works in the following way, first, the tracking detects a supposed pedestrian. This detection (the crop around the pedestrian) is then the input to our occlusion model (the focus of this work) which predicts the probability of the detection being an occluded pair instead of a single pedestrian. If the probability is above some threshold (this is done to minimize false positives and maximize precision) then the detection is counted two times, which reduces the error in the counting task.

We evaluated the model and the counting error was 111 out of 2251 passages (ground truth). The baseline without using our occlusion/pairs classifier had a counting error of 117. This means that using our classifier led to a decrease of 0.26% (calculated as $\frac{117-111}{2251} \times 100$) in the counting error. Those results are not good enough to be applied in iC systems.ai cameras yet, but they are promising for first results. Note that the decrease in the counting error can be partially caused by false negatives and false positives negating each other. Also note that in the future, we plan to improve our heuristic which determines whether a detection is a pair or not by working with the tracks in time. This adds valuable information about the previous frames, which are often the deciding factor even for human recognition.

Conclusion

The goal of this thesis was to first research existing work and methods related to the problem of occlusion detection, then to obtain suitable datasets and use them to train a pairs occlusion classifier. The final goal is to use this classifier to improve the performance of the existing iC Systems.ai tracking system in the task of counting pedestrians.

As our datasets are unclean, imbalanced, and not sufficiently big, we first discuss how those problems affect the performance of the model on the validation dataset and then offer solutions to those problems such as dataset cleaning and heuristics during the dataset creation process. After reaching a satisfying score in various evaluation metrics on the validation dataset, we evaluate how our pairs classifier improves the tracking system in the counting task. The resulting counting error is reduced from an error of 117 to an error of 111 out of 2251 pedestrian passages. This is a reduction of the counting error by 0.26% which is a promising result but not yet applicable for iC Systems.ai. Note, that the improvement is also likely in part caused by the negation of false positives and false negatives, which is a known problem with the evaluation of the counting task.

In the future, we plan to expand our training dataset which may improve the model a lot, because currently, the training dataset is still not big enough. We also plan to improve our heuristic determining if a detection is a pair or not with the addition of information about the previous detection. For example when a pair was present in the recent past then the single detection is more likely to be a pair in the present.

Bibliography

[1] C. Arteta, V. Lempitsky, J. A. Noble, and A. Zisserman, "Learning to detect partially overlapping instances", in *2013 IEEE Conference on Computer Vision and Pattern Recognition*, 2013, pp. 3230–3237. DOI: 10.1109/CVPR.2013.415.

[2] P. Kilambi, E. Ribnick, A. J. Joshi, O. Masoud, and N. Papanikolopoulos, "Estimating pedestrian counts in groups", *Computer Vision and Image Understanding*, vol. 110, no. 1, pp. 43–59, 2008, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2007.02.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207000392.

[3] W. Ge, R. T. Collins, and R. B. Ruback, "Vision-based analysis of small groups in pedestrian crowds", *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 34, no. 5, pp. 1003–1016, 2012. DOI: 10.1109/TPAMI.2011.176.

[4] M. C. Liem and D. M. Gavrila, "Joint multi-person detection and tracking from overlapping cameras", *Computer Vision and Image Understanding*, vol. 128, pp. 36–50, 2014, ISSN: 1077-3142. DOI: https://doi.org/10.1016/j.cviu.2014.06.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314214001301.

[5] M. Seise, S. Mckenna, I. Ricketts, and C. Wigderowitz, "Segmenting multiple objects with overlapping appearance and uncertainty", Jan. 2006, pp. 839–848. DOI: 10.5244/C.20.86.

[6] M.-C. Chen, "A video surveillance system designed to detect multiple falls", *Advances in Mechanical Engineering*, vol. 8, no. 4, p. 1 687 814 016 642 914, 2016. DOI: 10.1177/1687814016642914. eprint: https://doi.org/10.1177/1687814016642914. [Online]. Available: https://doi.org/10.1177/1687814016642914.

[7] C. Stauffer and W. Grimson, "Adaptive background mixture models for real-time tracking", in *Proceedings. 1999 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (Cat. No PR00149)*, vol. 2, 1999, 246–252 Vol. 2. DOI: 10.1109/CVPR.1999.784637.

[8] B. Ristic, "Detecting anomalies from a multitarget tracking output", *IEEE Transactions on Aerospace and Electronic Systems*, vol. 50, pp. 798–803, 2014.

[9] B. Romera-Paredes and P. H. S. Torr, "Recurrent instance segmentation", in *Computer Vision – ECCV 2016*, B. Leibe, J. Matas, N. Sebe, and M. Welling, Eds., Cham: Springer International Publishing, 2016, pp. 312–329, ISBN: 978-3-319-46466-4.

[10] C. Arteta, V. Lempitsky, J. A. Noble, and A. Zisserman, "Interactive object counting", in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds., Cham: Springer International Publishing, 2014, pp. 504–518, ISBN: 978-3-319-10578-9.

[11] R. Guerrero-Gómez-Olmedo, B. Torre-Jiménez, R. López-Sastre, S. Maldonado-Bascón, and D. Oñoro-Rubio, "Extremely overlapping vehicle counting", in *Pattern Recognition and Image Analysis*, R. Paredes, J. S. Cardoso, and X. M. Pardo, Eds., Cham: Springer International Publishing, 2015, pp. 423–431, ISBN: 978-3-319-19390-8.

[12] D. Kang, Z. Ma, and A. B. Chan, "Beyond counting: Comparisons of density maps for crowd analysis tasks - counting, detection, and tracking", *CoRR*, vol. abs/1705.10118, 2017. arXiv: 1705.10118. [Online]. Available: http://arxiv.org/abs/1705.10118.

[13] M. Schiegg, P. Hanslovsky, C. Haubold, U. Koethe, L. Hufnagel, and F. A. Hamprecht, "Graphical model for joint segmentation and tracking of multiple dividing cells", *Bioinformatics*, vol. 31, no. 6, pp. 948–956, Nov. 2014, ISSN: 1367-4803. DOI: 10.1093/bioinformatics/btu764. eprint: https://academic.oup.com/bioinformatics/article-pdf/31/6/948/17127270/btu764.pdf. [Online]. Available: https://doi.org/10.1093/bioinformatics/btu764.

[14] I. H. Laradji, N. Rostamzadeh, P. O. Pinheiro, D. Vazquez, and M. Schmidt, "Where are the blobs: Counting by localization with point supervision", in *Proceedings of the European Conference on Computer Vision (ECCV)*, Sep. 2018.

[15] Z. Ma, L. Yu, and A. B. Chan, "Small instance detection by integer programming on object density maps", in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 3689–3697. DOI: 10.1109/CVPR.2015.7298992.

[16] L. Wang, L. Xu, and M.-H. Yang, "Pedestrian detection in crowded scenes via scale and occlusion analysis", in *2016 IEEE International Conference on Image Processing (ICIP)*, 2016, pp. 1210–1214. DOI: 10.1109/ICIP.2016.7532550.

[17] S. Albawi, T. Abed Mohammed, and S. ALZAWI, "Understanding of a convolutional neural network", Aug. 2017. DOI: 10.1109/ICEngTechnol.2017.8308186.

[18] M. Xiang, "Convolutions: Transposed and deconvolution", 2020. [Online]. Available: https://medium.com/@marsxiang/convolutions-transposed-and-deconvolution-6430c358a5b6.

[19] S. Ruder, "An overview of multi-task learning in deep neural networks", *CoRR*, vol. abs/1706.05098, 2017. arXiv: 1706.05098. [Online]. Available: http://arxiv.org/abs/1706.05098.

Appendix A

List of used abbreviations

CNN Convolutional neural network

NLP Natural language processing

RNN Recurrent neural network

FCN Fully convolutional network

LSTM Long short-term memory network

ROI Region of interest

ReLU Rectified Linear Unit

Appendix B

Contents of the enclosed USB

Note, that as the codes are owned by iC Systems per the agreement with CTU FIT, they are not present on the enclosed USB but rather shared directly with the opponent of this thesis via GitLab for a limited time. Also note that the code in most of the files accessible in the GitLab repository is not written entirely by myself but rather expanded upon the work of my colleagues at iC Systems.ai. The files where most of my contribution is present are: pairs_pipeline.py, pedestrian_stroller_gender_pairs.py, Adaboost-OutputDatasetBboxRegression_pairs.py, clr_reg_gt_generator_NEW.py, eval_pairs.py, merge+create_datasets_script_NEW.py and pairs_heuristic_OUTPUT_is_frames+dataset.py. Most of those files are located in the path hydra/pipelines/necessary_files in the GitLab repository.

`thesis.pdf` ..... text of the thesis in the form of a PDF