Report for exercise 6 from group C
Comparing neural network and knowledge-based approaches for crowd dynamics

Tasks addressed:	6
Authors:	Alejandro Hernandez Artiles (03785345)
	Pavel Sindelar (03785154)
	Haoxiang Yang (03767758)
	Jianfeng Yue (03765255)
	Leonhard Chen (03711258)
Last compiled:	2024–03–02
Source code:	https://github.com/alejandrohdez00/Exercises-MLCMS-Group-C/tree/main/Exercise-6

The work on tasks was divided in the following way:

Alejandro Hernandez Artiles (03785345)	Task 1	25%
	Task 2	25%
	Task 3	25%
	Task 4	25%
	Task 5	25%
Pavel Sindelar (03785154)	Task 1	28%
	Task 2	28%
	Task 3	28%
	Task 4	28%
	Task 5	28%
Haoxiang Yang (03767758)	Task 1	1%
	Task 2	1%
	Task 3	1%
	Task 4	1%
	Task 5	1%
Jianfeng Yue (03765255)	Task 1	22%
	Task 2	22%
	Task 3	22%
	Task 4	22%
	Task 5	22%
Leonhard Chen (03711258)	Task 1	24%
	Task 2	24%
	Task 3	24%
	Task 4	24%
	Task 5	24%

Report on task 1, Introduction of approach and dataset (15%)

In this project we will implement, test and compare neural networks and knowledge-based approaches in the context of pedestrian dynamics.

The project is mainly based on two papers: *Prediction of Pedestrian Dynamics in Complex Architectures with Artificial Neural Networks* [10], where the authors use neural networks to predict the velocity of crowds in different scenarios and compare them to a simple knowledge-based approach, and *Review of Pedestrian Trajectory Prediction Methods: Comparing Deep Learning and Knowledge-based Approaches* [8], where an in-depth comparison between the two approaches is done, highlighting the pros and drawbacks of both of them.

In this task, we will explain the knowledge-based (with a focus in the speed-based model we are using) and the neural network approaches in more detail. Furthermore, we will define and review the dataset we will use to test and compare these models in a fair way.

Knowledge-based models Knowledge-based models are characterized by a defined set of rules or equations that take into account various factors such as physical, social, or psychological aspects of pedestrians. These models are typically described by a small number of parameters with clear physical interpretations, facilitating adjustments to the model.

Currently, knowledge-based models span macroscopic, mesoscopic, and microscopic scales, each addressing specific characteristics of the modeling scale. Macroscopic and mesoscopic approaches borrow concepts from continuous fluid dynamics or gas-kinetic models, providing an aggregated description of dynamics [5], [4]. On the other hand, microscopic approaches focus on modeling individual pedestrian motions. Researchers have extensively explored microscopic models, emphasizing the advantage of capturing heterogeneous behaviors due to their ability to individually represent pedestrians. This allows for the attribution of specific characteristics to each agent, considering behavioral heterogeneity and other diverse aspects.

In our previous projects and the current one, we have employed the microscopic approach within knowledge-based models. Therefore, our explanation from now on will focus on this particular subset of models.

These models, which describe individual pedestrian dynamics, are versatile in predicting pedestrian trajectories at any scale. By providing input information like the initial position, velocity, and acceleration of pedestrians, a forward simulation using these rules can forecast future trajectories.

The way the pedestrian motion to a new position is determined depends on the model's inputs and outputs. If the model outputs new velocity or acceleration, enabling the calculation of the new position, it is classified as a velocity- or acceleration-based model. Alternatively, if the position is directly determined by specific rules without involving differential equations, the models fall into the category known as decision-based (e.g. cellular automata).

Finally, we will focus on speed-based models, in particular, the one we will use in this project. Based on [10] we will implement and test a speed-based model to compare them to the neural networks. As in the paper, the goal is to predict the speed of pedestrians based on their positions and, if available, the velocity of their K closest neighbors. In the context of this description, (x, y) represents the position of the considered agent, v indicates its speed, while (x_i, y_i) for $i = 1, ..., K$ and (v_i, u_i) for $i = 1, ..., K$ denote the positions and velocities of the K closest neighbors, respectively.

The modelling approach is the parametric Weidmann fitting model [13]. In this model, the speed is represented as a non-linear function of the mean spacing, characterized by three parameters as:

$$W(\overline{s}_K, v_0, T, l) = v_0 \cdot \left(1 - e^{\frac{l-\overline{s}_K}{v_0 T}}\right) \tag{1}$$

$$\overline{s}_K = \frac{1}{K} \sum_i \sqrt{(x - x_i)^2 + (y - y_i)^2} \tag{2}$$

where $\overline{s}_K$ is the mean Euclidian spacing to the K closest neighbors. The time gap parameter T corresponds to the following time gap with the neighbors, v_0 is the speed of pedestrians in a free situation while l describes the physical size of a stopped pedestrian.

This model will be implemented and tested in Task 3.

Deep learning Deep learning involves training neural networks with more than two hidden layers. In this approach we rely on big amounts of data to learn the underlying dynamics of the pedestrians. In contrast to the knowledge-based approach, in the deep learning framework we do not need to design a set of rules or functions that aligns with the real dynamics of pedestrians, we obtain the function implicitly by training on the data of

past trajectories. Also, neural networks' parameters are non-interpretable (weights), i.e. they lack of a physical interpretation like in the Weidmann model [8].

In this section, we will present the basic neural network model that we plan to build in Task 2. Drawing inspiration from the approach outlined in [10], we intend to employ feed-forward artificial neural networks for predicting pedestrian speed. Our strategy involves experimenting with different architectures characterized by varying numbers of hidden layers denoted as h. In [10] they use 4 different types of combination of inputs parameters. However, we will use only the one that achieved the best results in the paper. This is because our project is more focused in comparing the two approaches, so we do not need the ones that performed badly. The neural networks are designed to produce predictions of pedestrian speed based on a given set of input parameters: The network we use depends on the relative positions and also the mean distance spacing $\bar{s}_K$ to the K closest neighbors (2K + 1 inputs):

$$NN_3 = NN_3(\bar{s}_K, (x_i - x, y_i - y, 1 \leq i \leq K))$$

This one corresponds to network definition NN_3 in [10].

We will see in Task 2 that it was difficult to replicate the results of this paper. Therefore, we divided our efforts: **Task 2 is divided into two parts**. One focusing in a data-centric approach, and the other one focusing on exploring different types of architectures.

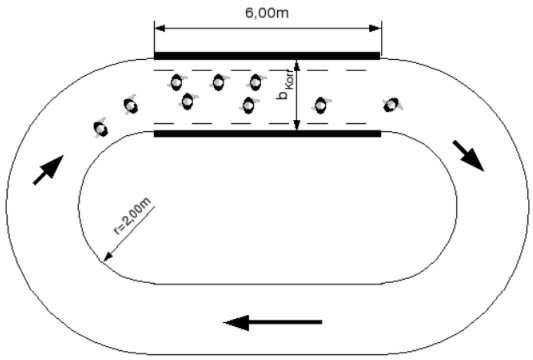
Dataset introduction To train, test and compare the models, we will use the dataset detailed in [6]. The dataset consists of trajectories of pedestrians in two different scenarios: The *Bottleneck* and the *Corridor*.

Independently of the scenario, each data row is composed by 5 columns:

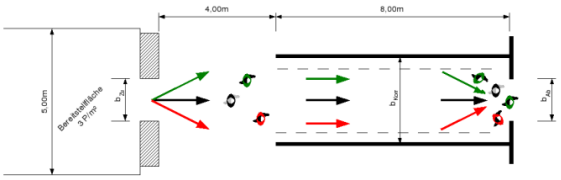
- ID of the pedestrian
- Frame number (frame rate is 1/16s)
- X coordinate
- Y coordinate
- Z coordinate

The corridor data comprises trajectories of pedestrians moving within a closed corridor with a length of 30m and a width of 1.8m. The trajectories are recorded over a specific section measuring 6m. The experiments involve varying participant numbers, specifically N=15, 30, 60, 85, 95, 110, 140, and 230. The experiments in this scenario are named with the prefix *UG*.

As for the bottleneck data, it consists of trajectories of pedestrians navigating through a bottleneck with a length of 8m and a width of 1.8m. The experiments in this case are conducted with 150 participants, exploring different bottleneck widths: w=0.7, 0.95, 1.2, and 1.8m. The experiments in this scenario are named with the prefix *UO*.



(a) Corridor experiment (UG)



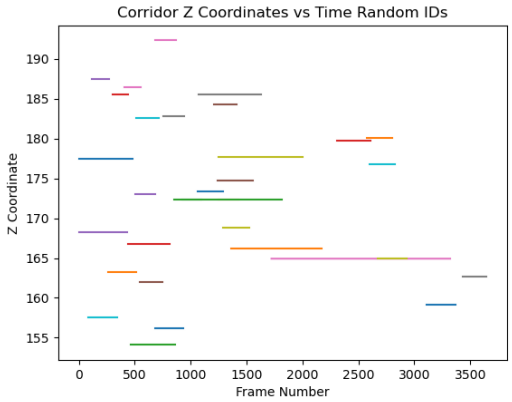
(b) Bottleneck experiment (UO)

Figure 1: Scheme of scenarios in the dataset

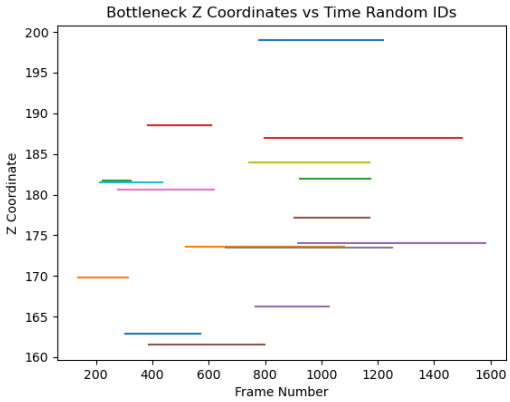
After providing an overview of the dataset, we proceed with an initial analysis. This analysis can be found in the notebook `data_visualization.ipynb`, that uses the functions in `utils/data_processing.py` and `utils/visualization.py`. The details about which code is related to each task can be easily found in the `README.md` in the repository.

Initially, it is important to note that the trajectories are three-dimensional, as the Z coordinate represents the height of each pedestrian. Considering the potential complexity introduced by the Z coordinate, we conduct tests to ensure if it indeed contributes relevant information.

Upon investigation, it was determined that the Z coordinate remains constant for each pedestrian throughout the observed time period. This can be visualized in Figure 2, where the Z coordinates of four randomly selected pedestrians are plotted over time for each experiment in both scenarios. Each colored line represents the Z coordinate for a specific ID in a given experiment.



(a) Corridor experiments Z coordinate

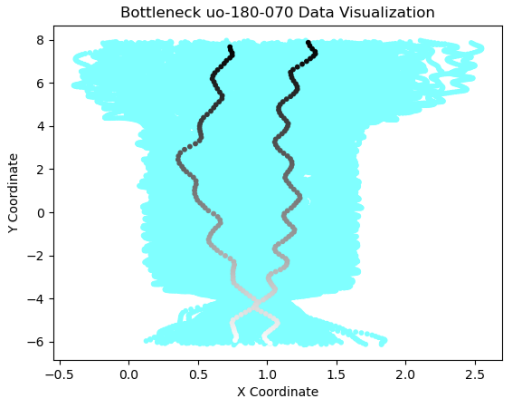


(b) Bottleneck experiments Z coordinate

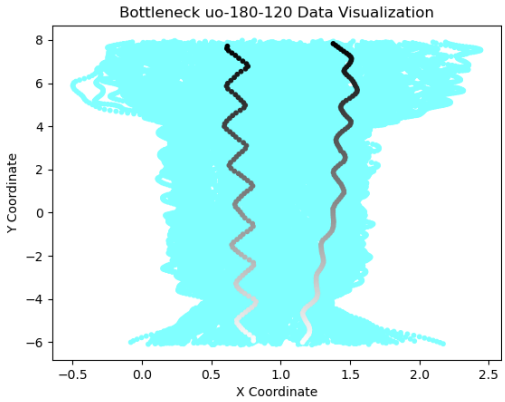
Figure 2: Z coordinate against the time

Due to this lack of change in the Z coordinate, we will not use it for our computations in the following tasks.

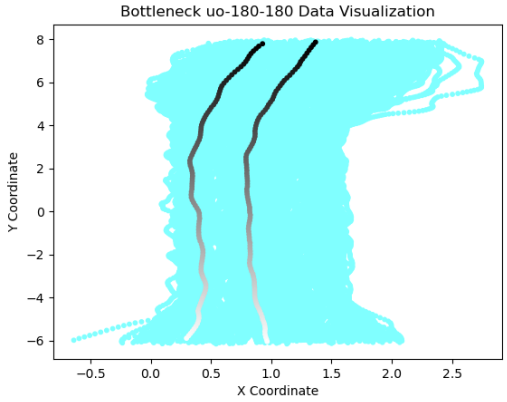
After this, we can visualize the trajectories data (i.e. the X and Y coordinates). In Figure 3 we can observe different experiments for each type of scenario. We highlight in black the two trajectories made for pedestrians with ID 1 and 2 for better understanding of the plot. The trajectories of the chosen pedestrians progress from black to white.



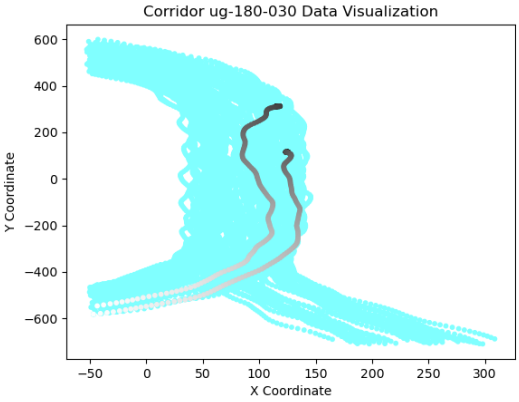
(a) Bottleneck experiment with width = 0.7m



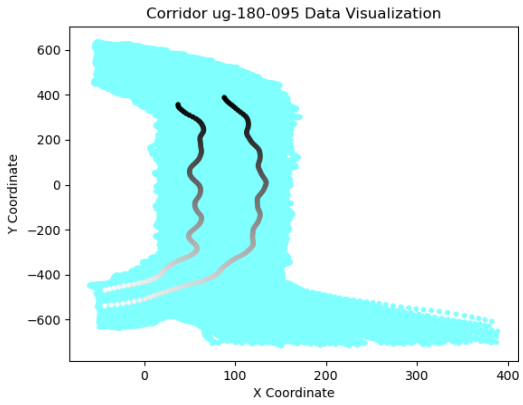
(b) Bottleneck experiment with width = 1.2m



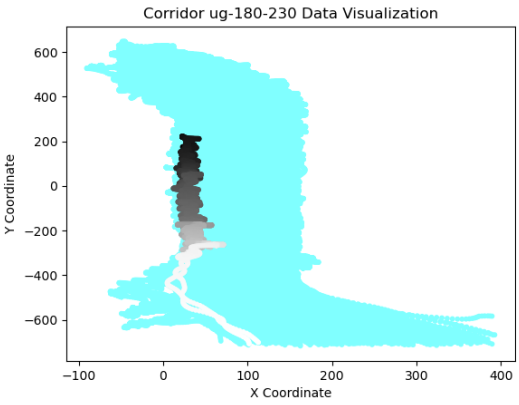
(c) Bottleneck experiment with width = 1.8m



(d) Corridor experiment with N = 30



(e) Corridor experiments with N = 95



(f) Corridor experiments with N = 230

Figure 3: Trajectories in different experiments

Comparing figures 3(a), 3(b) and 3(c), it is evident that the augmented bottleneck width, compared to the previous experiment, leads to trajectories being less closely spaced within the bottleneck. In the corridor scenario, a comparison of Figures 3(d), 3(e), and 3(f) shows that as the number of pedestrians increases, the number of trajectories also rises. Additionally, in Figure 3(f) the trajectories become more clustered and exhibit an apparent reduced speed, likely due to the higher volume of people in the corridor.

Report on task 2.1, Implementation of the neural network - Data focused approach (20%)

Data Preprocessing As stated before in the original dataset contains the following entries in its columns: ID, FRAME, X, Y, Z. On the other hand the paper uses a combination of relative positions, relative velocities, and mean spacing to predict speed. These training features and labels are not present in the experimental data therefore data preprocessing and it is required to handle these missing elements. This section describes how we preprocess the experimental data to generate training data.

According to the referenced paper and the experiments done in Task 1, the z coordinates of the velocity is not considered. Therefore, we have omitted the z coordinates.

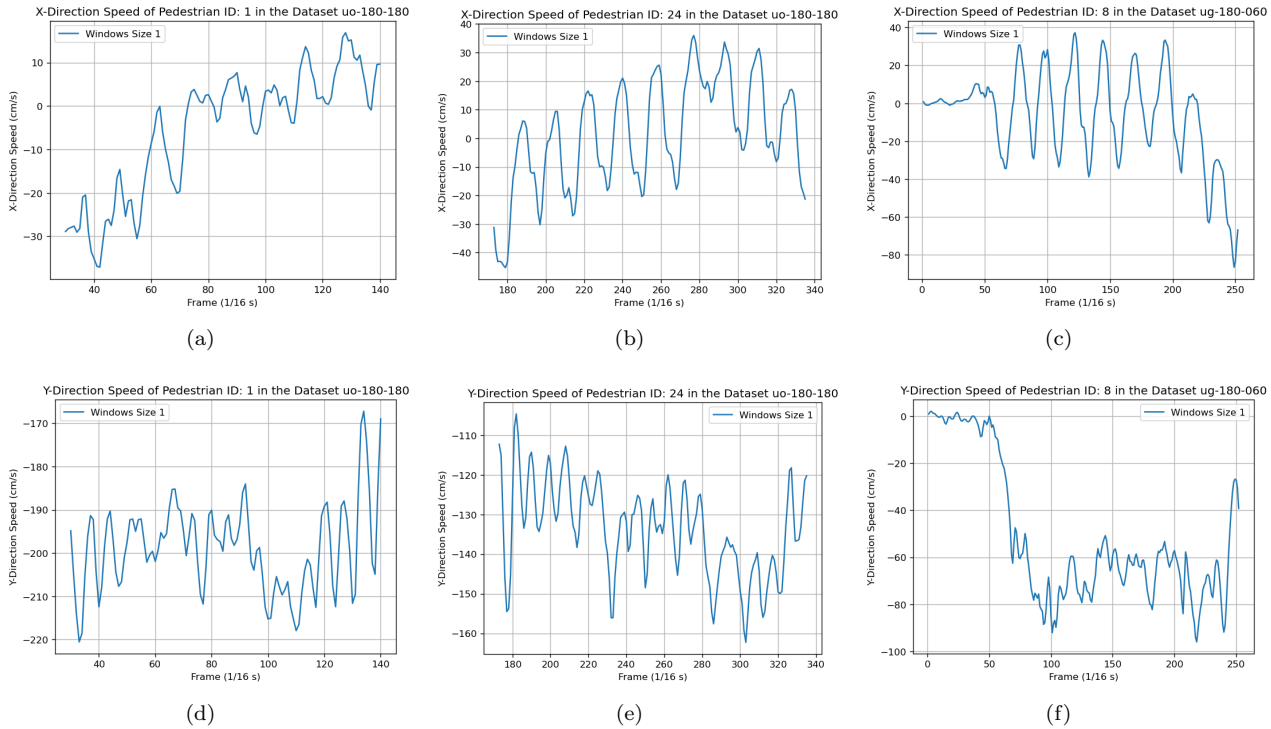
If we calculate the speed by dividing the position change from time t_i to t_{i-1} by the time difference, the resulting speed represents the average speed between t_{i-1} and t_i, which is not the speed at t_i. Therefore, we have decided to initially use a sliding window of size 1 to compute speed. This, i.e., averaging the speed between t_{i-1} and t_i with the speed between t_i and t_{i+1}, is expressed as:

$$\frac{\frac{x_i - x_{i-1}}{\Delta t} + \frac{x_{i+1} - x_i}{\Delta t}}{2}$$

where x_i represents the position at time t_i, and Δt is the time difference. Here, the partial x direction of the velocity is described, and a similar approach is applied to the y direction as well. Through this approach, for each data point, its speed cannot be calculated for the first frame and the last frame. Therefore, we exclude the first and last frames.

By training the network using data generated in this way, we found that the test loss of the trained model is significantly high. Consequently, we started considering further data processing.

In Figure 4, we illustrate the variation of the speed for pedestrian ID 1 and ID 24 in the uo-180-180 dataset and pedestrian ID 8 in the ug-180-060 over frames. Although the latter two are randomly selected, their behaviors are highly representative.



(a) (b) (c)

(d) (e) (f)

Figure 4: Speed of Pedestrians

By observing the figure, we can see that the pedestrian undergoes significant changes in velocity within one second. In the beginning, this change in velocity seemed very peculiar to us in some instances. For example there is nothing obstructing the first pedestrian so theoretically his velocity shouldn't be fluctuating. If the

change in velocity is not influenced by external factors, then the reason for the velocity variation can only come from the individual itself. After reviewing a large number of pedestrians in the database, we found that almost every pedestrian exhibits similar velocity variations. This situation has let us to start considering the process of walking motions.

If people's walking speed variation follows a smooth curve, they should appear as if gliding on a skateboard rather than walking in a normal manner. Taking this into consideration, coupled with the fact that the data capture point are on the tester's head, we found the reason for the speed variation—when people walk, the alternating use of left and right feet causes lateral speed changes. Speed slows down when the foot is extended and accelerates when taking a step forward, resulting in longitudinal speed variations.

However, these speed changes, caused by these movement motion, are not related to avoiding pedestrians. These circumstances lead to a significant amount of noise in the data, making it challenging for our simple network to capture patterns of avoiding pedestrians. This is the reason for the high test loss.

In the Bottleneck Data dataset, we attempted to set the x-direction velocity to 0, assuming pedestrians wouldn't sway left or right, surprisingly achieving a smaller test loss. Therefore, in further data preprocessing, we intend to minimize the impact of walking motions and focus on the influence of pedestrians on the speeds of other pedestrians.

Then, we tried a larger sliding window. Figure 5 compares the velocities generated using sliding windows of sizes 1, 8, and 16. The generated speed correspond to average speeds within 2/16 seconds, 16/16 seconds, and 32/16 seconds, respectively.

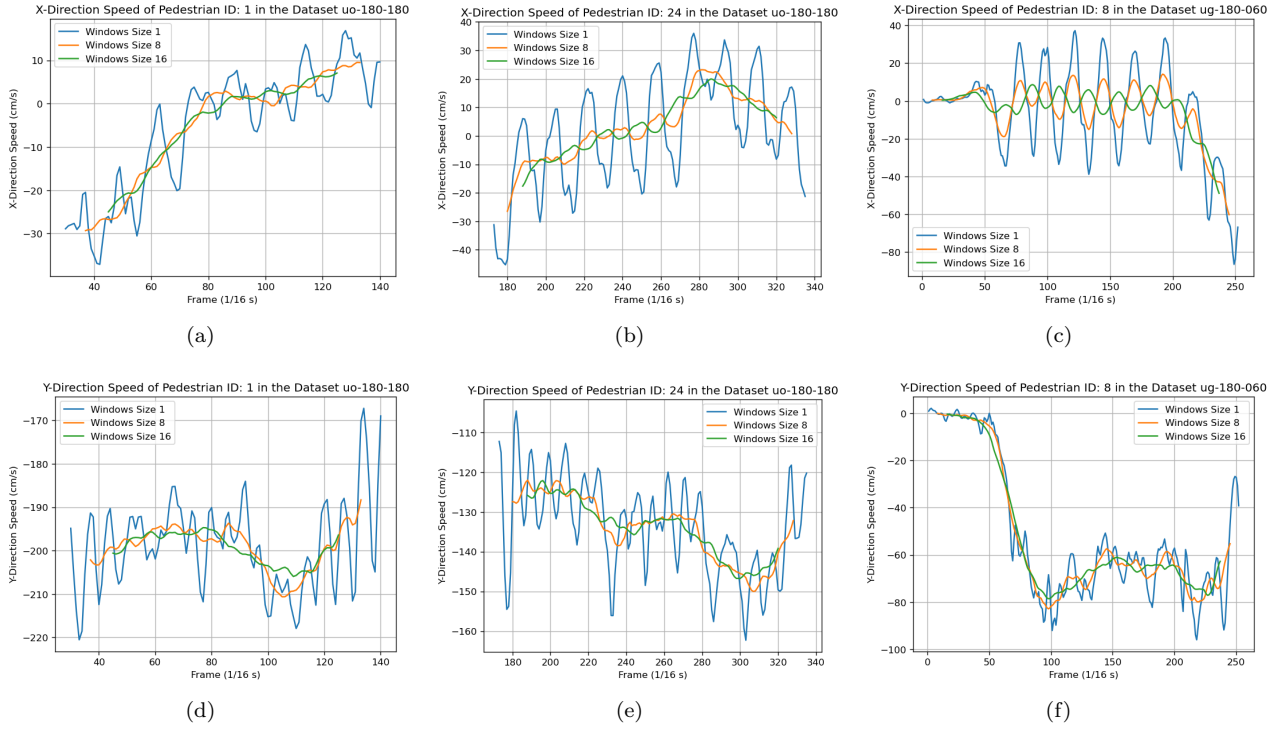


Figure 5: Speed of Pedestrians

From the graph, it can be observed that even with a very large window size, the speed in x direction still exhibits fluctuations. Other pedestrians are not running and are unlikely to appear instantly beside the current pedestrian, so the speed variation should be smoother. This leads us to consider another method for handling the x direction of velocity. Because speed is the distance covered by pedestrians in a unit of time, we started considering whether the path of pedestrians is smooth.

Figure 6 illustrates the trajectory changes in the x and y directions for pedestrian ID 1 and ID 24 and pedestrian ID 8 in the ug-180-060, respectively.

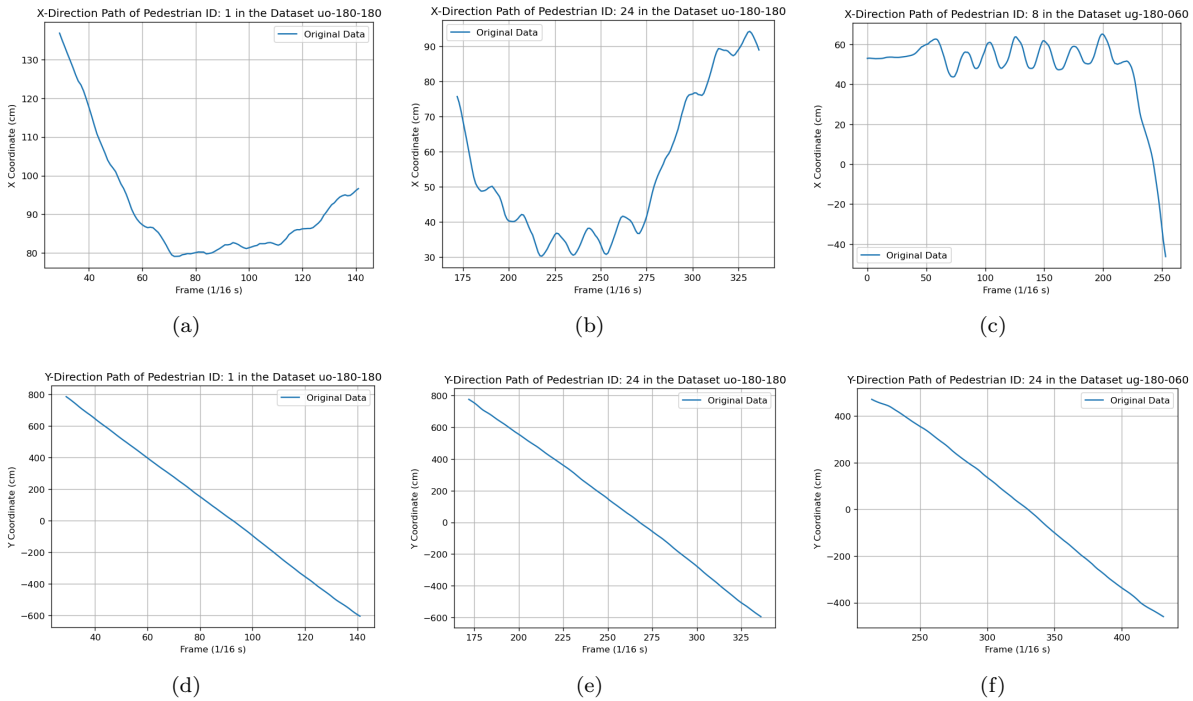


Figure 6: Path of Pedestrians

The behavior of most other pedestrians in the database is similar to the showcased pedestrians. As observed, while the trajectory is very smooth in the y-direction in both databases, there are fluctuations in the x-direction. Especially when trying to maintain the x-value, there is significant fluctuation in x. In other words, when pedestrians are walking straight forward, there is lateral swinging. To eliminate the impact of pedestrians' swinging on the data, we can fit a polynomial function to the trajectory. While the trajectory in the y-direction is smooth, no processing is needed.

Figure 7 shows the x-trajectory curve fitted with a polynomial function of degree 6.

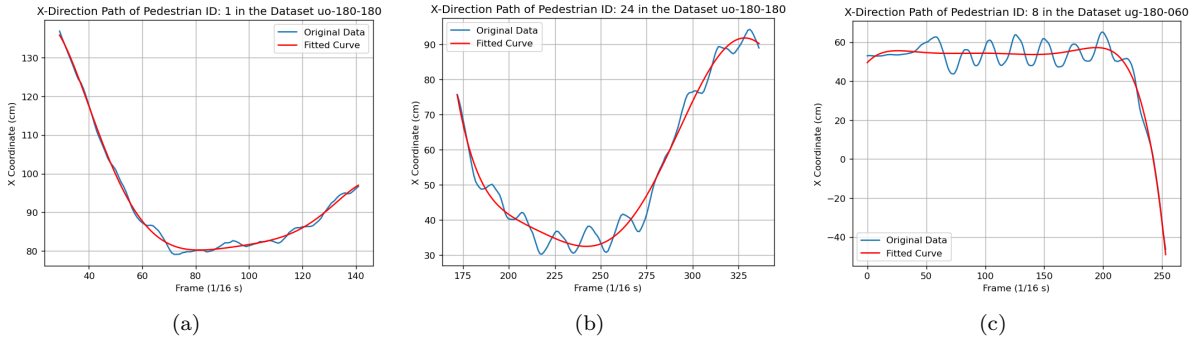
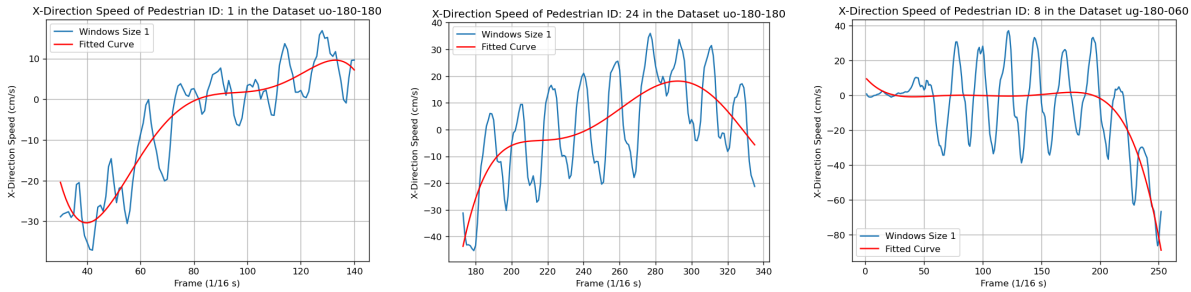


Figure 7: Fitted Path of Pedestrians in x Direction

After this processing, the trajectory becomes significantly smoother, offering a seemingly more accurate representation of pedestrian paths. Simultaneously, it's worth noting, as observed in Figure 7(c), that this processing method has minimal impact on the trajectories when pedestrians make turning movements.

Using the fitted curve as the new x-direction trajectory, and then calculating the x direction speed with a sliding window of size 1, the speed curve is shown in Figure 8.



(a) (b) (c)

Figure 8: Fitted Speed of Pedestrians in x Direction

For the y-direction, since the path is very smooth and the sliding window method performs very well, there is no need to use this method. Therefore, a sliding window with a size of 8 was employed.

In this approach, We set the value of k, representing the number of neighbors, to 10. when the number of people around a pedestrian is less than 10, such data is not considered because S_k as well as $(x_i - x, y_i - y, v_i - v, u_i - u, 1 \leq i \leq K)$ could be inaccurately estimated. However, each individual's contribution to velocity calculation is not equal among the 10 neighbors. The influence of the closest neighbors on pedestrian velocity is expected to be more significant. Therefore, during the data preprocessing stage, we sorted pedestrians from the nearest to the farthest. This ensures that data for x_0, y_0, v_0, u_0 always originates from the closest pedestrian, while data for x_9, y_9, v_9, u_9 always comes from the farthest pedestrian. Features are sorted based on distance and then input to the corresponding neurons in the network. This facilitates the model in learning patterns related to the proximity of pedestrians more effectively.

Thus, we have acquired the data points utilized for training the neural network. From the Bottleneck and Corridor datasets, we obtained 240,186 and 791,023 data points, respectively.

Network Training In our initial experiments, we attempted to use the same neural network architecture as described in the paper. Let a and b denote the number of neurons in the first and second layers, respectively. Define the hidden layers as $h = (a, b)$ (or simply $h = (a)$ if $b = 0$). The evaluated hidden layers h consist of (2), (3), $(4, 2)$, $(5, 2)$, $(5, 3)$, $(6, 3)$, and $(10, 4)$. We chose to only train the NN4 network with an input size of 4*K+1, as it demands the maximum number of input features. This is more advantageous for uncovering differences before and after applying our data preprocessing method.

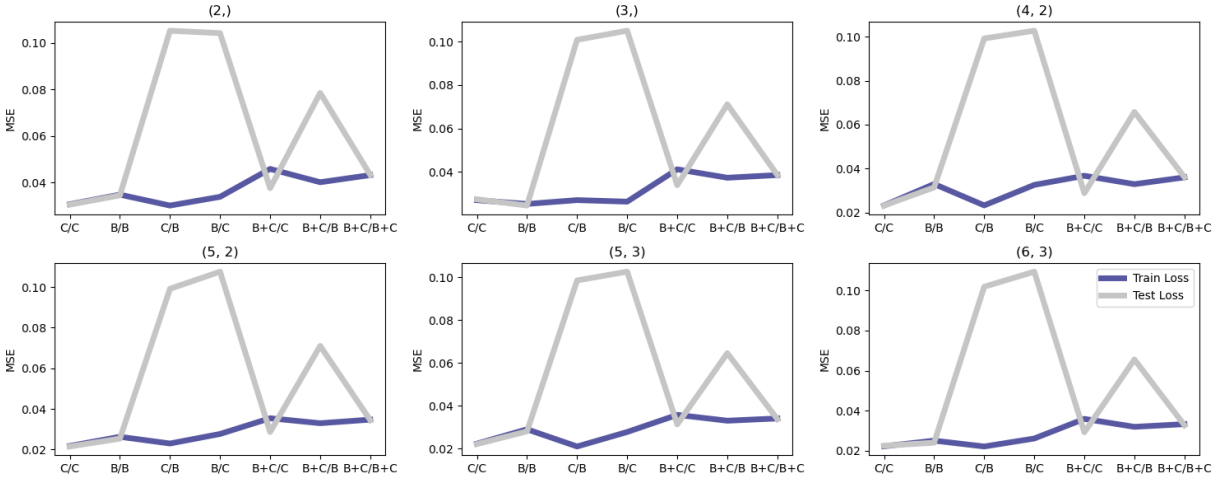
In the upcoming sections, the initial parameter X in the notation 'X/Y' indicates the dataset utilized during the training phase, with the subsequent parameter Y representing the dataset employed for the testing phase.

For B/B, C/C, and C+B/C+B, both training and testing are conducted. The data is partitioned into a 6:2:2 ratio, serving as the training set, validation set, and test set. For B/C and C/B, these scenarios are designed to assess the model's predictive capability in novel situations. The training data is split into an 8:2 ratio for the training set and validation set, with the other dataset employed as the test set. As for C+B/B and C+B/C, initially, the dataset is divided into a 7:3 ratio, with 3 reserved for testing. The remaining 7 are combined with another dataset and further divided into an 8:2 ratio, with 2 allocated for the validation set.

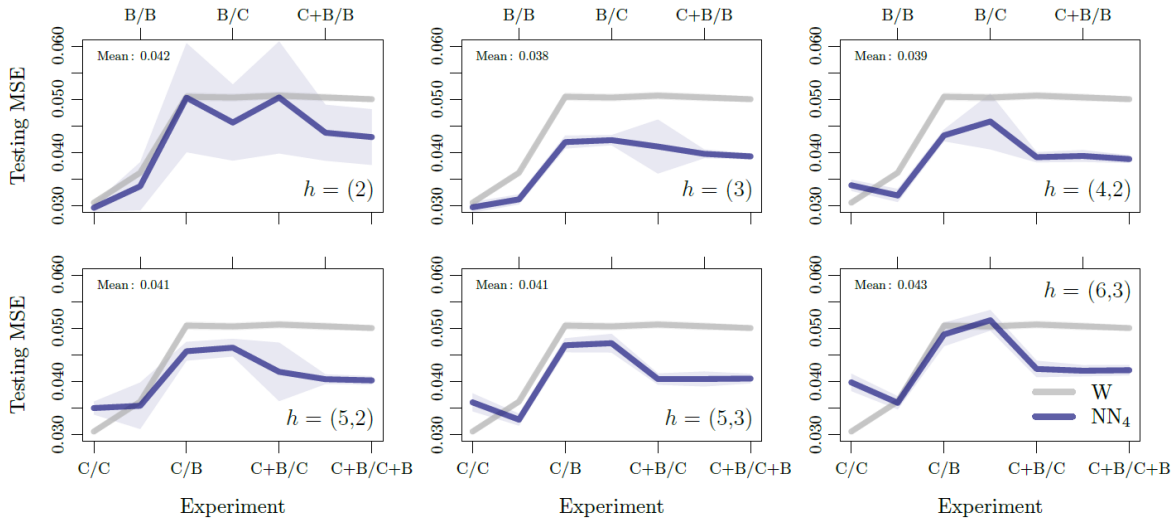
Before inputting the data into the network, we use the torch dataloader's shuffle function to randomize the training data sequence and set the batch size to 32. We employed Mean Squared Error (MSE) as the loss function. The optimizer utilized for training is Adam with a learning rate of 0.001.

During the training of the network, we observed that a simple network converges rapidly, stabilizing within the two epochs. This could be attributed to the abundance of data. Because we need to test six different networks across seven data scenarios, the total number of networks to be trained amounts to 56. Therefore, we have to use a relatively small number of epochs to save time. The training epoch for all networks is set to 3.

Result Due to time and computing limitations and the abundance of data, cross-validation was not implemented in this section. The network underwent training only once. The results can be observed in Figure 9.



(a)



(b)

Figure 9: Result

The upper portion of the figures is generated from our result, while the lower part is from the paper. It can be observed that in certain scenarios, our performance surpasses that of the paper, while in others, it falls short. However, we only trained for 3 epochs. If each network could be trained until the validation loss starts to rise, it is highly likely that the performance in scenarios other than C/B, B/C, and C+B/B would significantly outperform the paper. The sub-optimal performance on C/B and B/C scenarios may be attributed to the removal of noise, which we suspect leads to a slight loss of information, hindering the model's generalization to unknown datasets.

The model's performance in the C+B/B scenario is not satisfactory, and we suspect it is due to data imbalance. It might be beneficial to discard a portion of the excessive C data, achieving a balance by training the network with an equal quantity of B and C data. With this mindset, we utilized the (5,3) architecture and conducted training for 20 epochs on C+B/B with an equal number of data points for C and B. The resulting test loss was 0.063. However, this was hardly any improvement compared to the previous attempt. This did not align with our expectations. Therefore, we trained with the same architecture and epoch count without discarding any data points, yielding a test loss of 0.056. While there was some improvement, it still falls short of the results in the paper. We can roughly conclude that our data preprocessing method has had a certain impact on the generalization ability of the trained model.

During the experimentation process, we observed that the network described in the paper appears to be too simplistic, posing challenges in capturing intricate patterns. Even when attempting to overfit a simple network (3,) with a minimal number of data points, such as 500 data points, achieving a loss close to zero proves difficult. This implies limitations in the model's memorization capabilities. Consequently, we experimented with a significantly larger network (256,128,64,32) on C+B/C+B with 60 epochs, and the obtained test loss is 0.007. And on B/B with 50 epochs, the test loss is 0.006. This confirms our speculation that choosing larger models could result in more precise speed predictions.

Future Considerations In the existing dataset, the pedestrian movement is relatively uni-direcrtional. Perhaps in subsequent attempts, data augmentation techniques could be employed. For instance, rotating the coordinates around any angle axis could cause pedestrians to face different directions, significantly increasing the number of data points.

Due to the varying impact of nearby pedestrians at different distances, with the closest pedestrians having the most significant influence on the current pedestrian, it is challenging for the simple network we are using to identify this. Introducing an attention mechanism may help the model focus its attention on pedestrians in very close proximity. This, at the very least, could result in faster convergence speed for the model and potentially even lead to better results.

Conclusion Our Data preprocessing method can eliminate noise from the raw data, making it easier for the model to learn patterns. This not only leads to faster convergence of the loss function but also results in a smaller test loss. However, it may not necessarily contribute to enhancing the model's ability to generalize predictions on unknown datasets.

Report on task 2.2, Implementation of the neural network - Architecture Focus approach (20%)

Architectural focus The following results were completed before the first part of this task and we only later found out decreasing the batch size significantly together with removing unnecessary pedestrians and taking every frame allows us to recreate the results from the paper. This part of the task therefore mostly concerns itself with working in the context of mega-batches and an 80 frame sampling period.

Training procedure For the training we try the same procedure as the paper [10]. For a given experiment taking different combinations of experiments. We take either all the data from an experiment type if it is only present in one split (B/C, C/B). In cases where it is present in multiple splits we do a 50/50 train test split and adjust the training set so it is roughly balanced. We always use the whole test set in a batch.

Afterwards we run k-fold bootstrapping (like cross-validation except we only use it to select the best model) on the train set to find the best parameters. We use all the datapoints in each batch because there are only tens of thousands of them.

The data in the paper is processed by taking one datapoint every 80 frames and getting the speed/ velocity by taking a point 16 frames in the future. We can better approximate the speed by taking the average of the speed in the eight previous and the eight next frames. We use one datapoint every 80 frames but decreasing the spacing does not impact model performance on the generalization experiments.

In the paper they also talk about getting the Weidmann parameter's from the neural networks which is at odds with what the rest of their paper seems to say. We tried instead of directly predicting the speed, to predict the Weidmann model parameters and use Weidmann's equation to create predictions but this has abysmal results with any hyperparameters we could find. Another thing we tried was to predict 2 values and then take $x * exp(y)$. Both has higher or equal loss so in the experiments shown we only use the network with no activation at the end.

Trying to recreate the original results We were unable to recreate the original results even when using the original data processing in 10. We only tried the NN3 architecture from the paper (it uses relative positions of the k nearest pedestrians and their mean spacing) as it has the best results. The paper did not mention whether the pedestrians that do not have enough neighbors should be removed so we assume they should not be (if they have at least two neighbors) and instead replace the values of these neighbors with zero and only calculate mean spacing from the neighbors they do have. When we later tried to remove the pedestrians without enough neighbors the model performance did not significantly change. For the number of neighbors we follow the example of the paper and use 10.

We use a weight decay of 0.2, learning rate of 1e-3, learning rate decay of 0.98 and run the training for 50 epochs. Later we also trying these architectures on our data 11. We can see that the models produced using the method from the paper have comparable results.

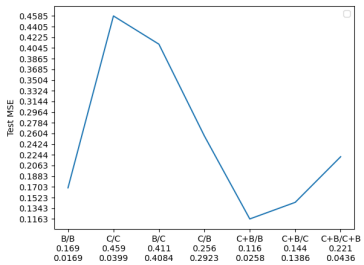
Off the shelf second order optimization Since we are already using a massive fraction of the dataset each iteration the idea came up that we could simply use the whole dataset each iteration and use a second order method.

These are methods similar to the newton method and use the first and second derivatives. This is normally very unfortunate because it means we need p^2 space and p^3 time where p is the usually very large number of parameters and of course we need to fit all the data and copies of the data for paramater purposes in the memory along with this.

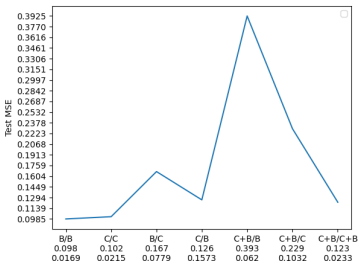
The specific method we use is described in [9]. In the paper they say that while their method is derived in a different way than the newton method it can be seen as a generalization of it. The paper also talks about how important weight regularization can be for second order methods and seeing as our methods seem to fail to generalize that well these might be good to use. Because the convergence of these methods is quite slow we only attempt them on the NN4 and layer sizes [(3,), (4,2,), (5,2,), (5,3,), (6,3,), (10,4,)] and the C+B/C+B dataset.

We use no weight decay, again a learning rate of 6e-2, learning rate decay of 0.98 and run the training for 50 epochs.

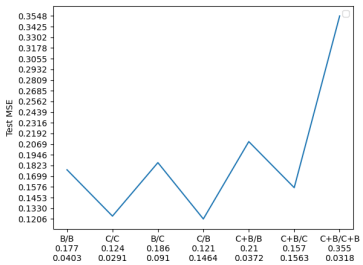
As we can observe in Figure 12 our networks ouperform their architectures on this data with our parameters. Especially the large networks seem to scale better. This lead to the understandable desire to see how even larger networks would perform.



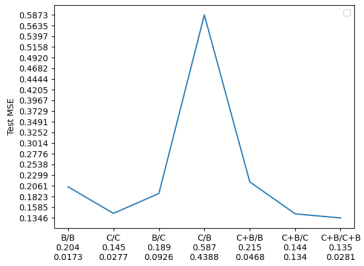
(a) (1,)



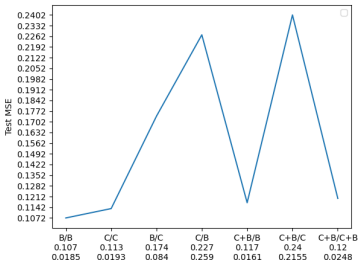
(b) (2,)



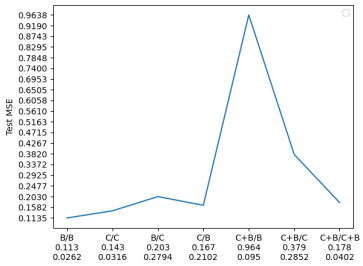
(c) (3,)



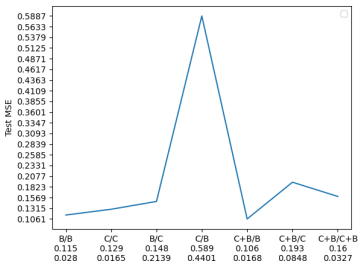
(d) (5,2)



(e) (5,3)

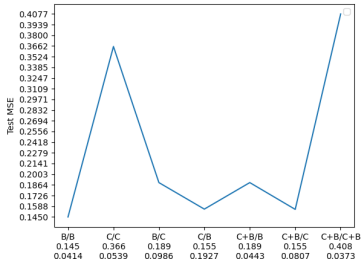


(f) (6,3)

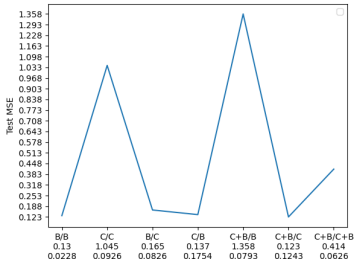


(g) (10,4)

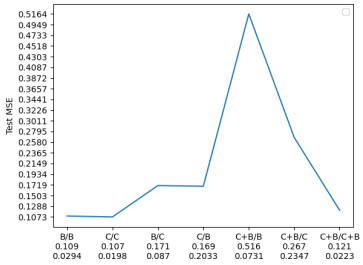
Figure 10: Every 80 frames looking 16 frames into the future using the NN3 architecture



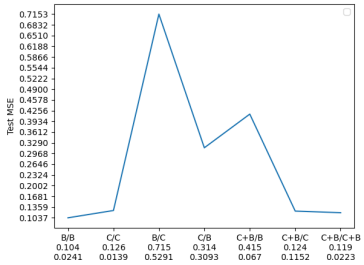
(a) (1,)



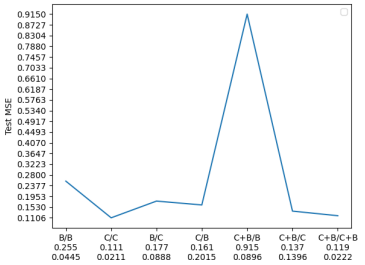
(b) (2,)



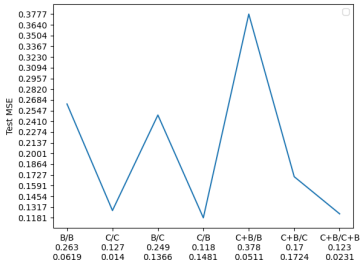
(c) (3,)



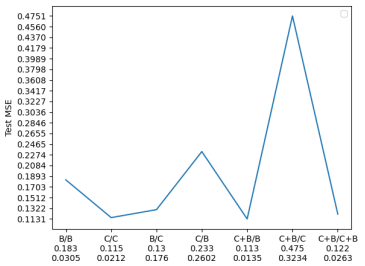
(d) (5,2)



(e) (5,3)

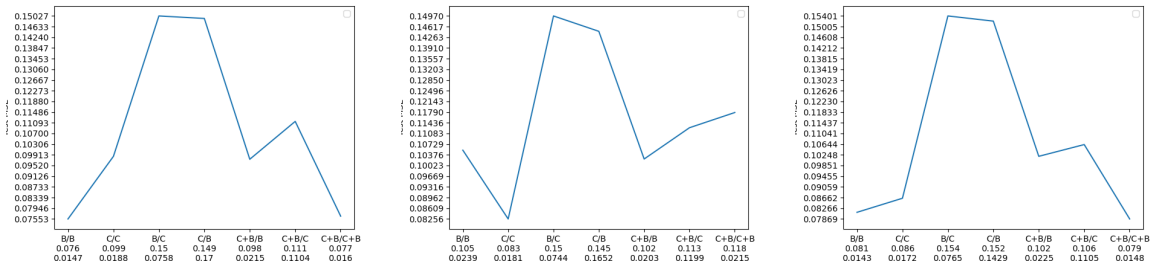


(f) (6,3)



(g) (10,4)

Figure 11: Every 80 frames taking average speed in 8 frame window behind and forward in time the NN3 architecture



(a) (3,) (b) (5,2) (c) (5,3)

(d) (6,3) (e) (10,4)

Figure 12: Every 80 frames taking average speed in 8 frame window behind and forward in time using the NN3 architecture and a second order optimizer with default arguements

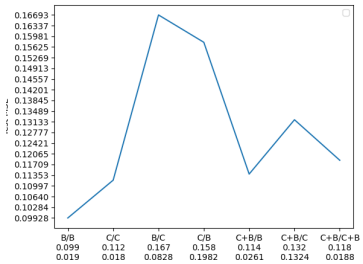


Figure 13: large network (12,5,3), second order optimization, regularization and less momentum

Second order parameter optimization and regularization We tried large networks, the MSE for the very simple C/C and B/B scenarios was even better but they generalized even worse.

To remedy this we try to use different optimizer parameters, most interesting is the regularization parameter and the momentum parameter. We also add our own L2 regularization directly to the loss as the method we use lacks it. We use all different parameters for each experiment and select the best network using the validation loss.

Eventually after increasing the number of parameters to (12,5,3) and the number of epochs up to 100 we get the following results. We use no weight decay, learning rate of 6e-2, learning rate decay of 0.98 and run the training. We are accumulating over up to 30 iterations (we still step in each iteration we just use all 30 last iterations).

As we see in Figure 13 there appears to be no significant change. This does not mean that there might not be a combination of parameters we did not try that wouldn't yield better results but because of time constraints we did not find it.

Variation method To try to further increase generalization we try to use a variational architecture. Instead of doing just fully connected layers we use $l * l + l$ neurons to create the variance matrix and the mean for the

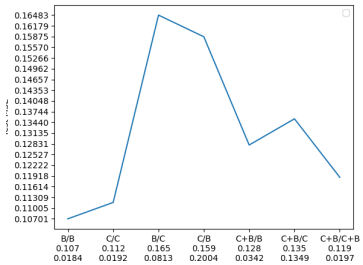


Figure 14: variational network second order optimization l = 4

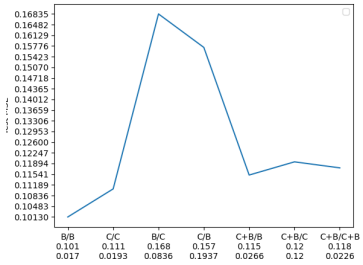


Figure 15: small network (5,5), second order optimization regularization, minumal momentum and regularizing using the knowledge based model

gaussian. This seems to have more potential to fit the data but generalizes even worse 14. One interesting thing is that while the resulting model has more parameters it also requires less training.

Hybrid method To help regularize the model we make it resemble the Weidmann model more (ie. we remove the last relu activation and use the exp function). We also use a Weidmann model whose parameters are optimized on the training data and use the distance to it's output with half the weight decay. While it seemed to make sense to use gradient descent here also to get as good an estimate as possible using a simple grid search to get the Weidmann model parameters seems to also produce a good model for the purposes of regularizing the training of our network. It also seems that when we remove the momentum from our model a much smaller network size presents itself.

We can see in Figure 15 that the hybrid model as we implemented it did not significantly improve performance. When we look at the experiments done on the original data we see that it outperforms the models in Figure 16 but when we take away the knowledge based regularization (Figure 17) it seems that it was not the determining factor.

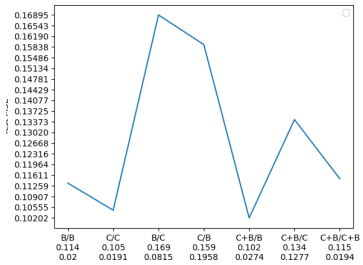


Figure 16: small network (5,5), second order optimization regularization, minimal momentum and regularizing using the knowledge based model evaluated on the data prepared with the procedure from the paper

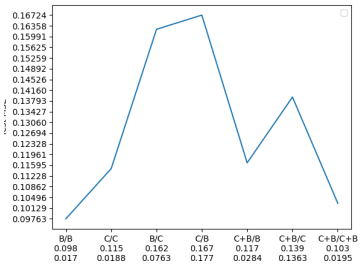


Figure 17: small network (5,5), second order optimization regularization, minimal momentum evaluated on the data prepared with the procedure from the paper

Conclusion It seems with the selected data processing methods the choice of architecture did not matter much. However the second order optimization seems to have improved the loss by a small margin. In spite of having the more or less the same relative results between models, our losses for the recreation of the conditions in the paper are significantly higher than the ones in the paper. It is difficult to know why exactly, as the paper did not publish any of their training details like number of epochs, learning rate or weight decay.

Report on task 3, Implementation of knowledge-based model (20%)

In this task our goal is to implement the knowledge-based model mentioned in paper [10]. Specifically this model is the parametric Weidmann fitting model for the fundamental diagram. This is a microscopic speed-based model and hence the speed of a single pedestrian is modeled. Before we start with the implementation, we want to provide a brief primer on fundamental diagrams.

In the context of traffic flow, a fundamental diagram graphically illustrates the relationship between flow, density, and speed. It is used to analyze and understand the behavior of traffic in different conditions. In our case of the Weidmann model it is used to model pedestrian speed and from here on we can draw conclusions on pedestrian flow.

Looking into the original document [13] by Weidmann from 1993, we were not able to find the function provided in this paper [10]. But on page 62 section 4.3.1 "Leistungsfähigkeit in der Ebene" of the original document [13] we did find a similar expression that describes pedestrian speed in a similar non-linear way.

$$v_{F,hi} = v_{F,hf} \cdot \left(1 - e^{-\gamma \cdot (\frac{1}{D} - \frac{1}{D_{max}})}\right) \tag{3}$$

Here in 3 $v_{F,hi}$ is the speed under a certain density, while $v_{F,hf}$ is the speed under a free situation. Next D describes the current density of the pedestrian, while D_{max} is the maximum density, such that a pedestrian can not move anymore. Additionally there is the calibration constant γ, which for example would have a different value, when considering pedestrian speeds for stairways.

The Weidmann model Now that we have seen the original Weidmann model3 it turns out that the model presented by Tordeux[10] differs from it. The original document uses the pedestrian density D as function parameter, while in this paper[10] the mean spacing $\overline{s}_K$ is used. Then, the model is defined by equations 1, 2.

Recall that the variables are described as follows: Here $\overline{s}_K$ is the mean spacing of the pedestrian to its K closest neighbours. v_0 is the speed in a free situation. l is the physical size of the pedestrian and T describes the following time gap with the neighbours. Once again we have to critize the authors as the variables l and T are not further detailed in the paper and it is up to speculation on how they are precisely defined. Furthermore it is also unknown how this Weidmann model was obtained from original expression 3, though at least it can be easily verfied that the units of the variables are correct.

In the original model the variables γ, D, D_{max} have units (P/m^2), which cancel each other out. Similarly the variables $l, \overline{s}_K$ are in units (m) and cancel out with v_0 (m/s) and T (s). In total only m/s remains from v_0 and $v_{F,hf}$ in the functions $W(...)$ and $v_{F,hi}$, and both describe a speed.

Fitting the parameters for the Weidmann model As stated previously our goal is to fit the parameters of the Weidmann model to the dataset used in the paper[10]. By fitting the parameters we will be able to use the function $W(...)$ to plot the fundamental diagrams. The difficult part is to find all parameters used by the function 1, so that we can estimate the function. As stated in the paper [10] section 3.3 the function for the fundamental diagram was obtained after performing least squares fitting using the data on the Weidmann model.

Data preprocessing Given in our raw dataset we have pedestrian ids, frame ids and (x, y, z) coordinates. In our python notebook `preprocessing.ipynb` we first perform some simple data cleaning, which after cleaning are stored in the folders `Bottleneck_Clean` and `Corridor_Clean`. Here the `transform(...)` function from `utils.data_processing.py` removes the z-coordinate, as it serves no purpose and additionally x, y-coordinates are scaled to meters from cm.

Next the functions `remove_bottleneck_oob(...)` and `remove_corridor_oob(...)` remove x, y-coordinates that are out of bounds for each of the test scenarios. While this step ins't detailed in the paper, we know from the paper that measurements are taken within a well defined area as well as that the x, y-coordinates are provided in cm. From here we conclude that we can remove coordinates that lie outside of the specified measurement area. For the "Bottleneck-Scenario" we removed any coordinate that lies outside of the boundaries for $x \in (0.0, 8.0)$ and $y \in (0.0, 1.8)$. Similarly we removed coordinates from the "Corridor-Scenario" whenever the lie outside of $x \in (0.0, 6.0)$ and $y \in (0.0, 1.8)$. With the basic data cleaning done we are going to calculate the mean spacing $\overline{s}_K$, speeds v and density D. For each pedestrian at each frame. Since the data preprocessing isn't detailed in the paper[10], we tried to follow as closely and reasonably as possible to the paper, though we were not able to achieve the exact same results.

Calculating $\overline{s}_K$ For the calculation of the mean spacing we used $K = 10$ neighbours, since the paper states that $K = 10$ was the highest reasonable value. The paper does not specify how it treats cases when there are less than 10 nearest neighbours. Here we opted to simply calculate the mean spacing $\overline{s}_K$ using only the available neighbours, if there were any.

In table 1 we see the mean value of our calculated mean spacing as well as the standard deviation. Our result differ from the paper[10], but we can also see a similarity, that the bottleneck scenario has a higher mean than the corridor scenario. This is expected, since their data preprocessing is unknown.

Scenario	mean (our)	mean (paper)
Bottleneck	0.910 ± 0.272	1.14 ± 0.37
Corridor	0.905 ± 0.330	1.03 ± 0.40

Table 1: Mean value of spacing

Calculating v For the calculation the speed v we tried 2 different approaches.

- In our initial approach (old) we calculate the speed by using the previous frame. This means the speed of a pedestrian during a specific frame is the traveled distance since the last frame. Formally we calculate the euclidean distance between the current position and position of the previous frame. In the special case of the first frame we assume that the pedestrian did not move, hence we set the previous frame position to be the same as first frame position. Bu this method is flawed as calculating speeds on a per frame basis (1/16 of a second) and scaling them up by $\times 16$ leads to inaccuracies.
- In our main approach we instead follow the approach from the paper, were speeds are obtained by differentiating positions over a second (see [10] section 3). In terms of implementation, this means we have to consider 16 frames to calculate speeds for a single frame. This leads to the special case where the first and last 8 frames are undefined. We decided here to instead use the average spead of this pedestrian after calculating the speeds for each frame. Additionally in the case where we have less than 16 frames, we use the initial approach to calculate the speed.

In table 2 we see the mean value of our calculated speeds. Here we can see a comparison between our two approaches, the initial one being marked as old, and the results from the paper [10]. As explained in the previous section our data preprocessing probably differs from the one in the paper and therefore our results are slightly different as expected. But we can also see similarities once again, as the mean speed in the bottleneck scenario is much higher than in the corridor scenario. We also have closer values for the standard deviation in our second approach. The old approach calculated speed over a single frame (1/16s), which causes some speeds to have very high values. This is also reflected in the stndard deviation of the old approach. From those 2 approaches we also learn that the corridor scenario has a much lower speed overall.

Scenario	mean (our)	mean (paper)	mean (our - old)
Bottleneck	0.554 ± 0.324	0.72 ± 0.34	0.623 ± 0.715
Corridor	0.327 ± 0.305	0.35 ± 0.33	0.375 ± 0.332

Table 2: Mean value of speed

Calculating D Finally we also decided to calculate the densities D for each frame, though it is expanded to each pedestrian, so it can be assigned to each of them in the dataset. For the density we simply counted the amount of people that are within the measurement area and divided that count by the measurement area. In conclusion the mean spacing $\overline{s}_K$, speed v and density D can all be calculated from the data. These are then appended to the cleaned data, which then form our new cleaned dataset.

In table 3 we can see our estimations for density in person per m^2. Here we see that the bottleneck scenario overall has a higher mean for pedestrian density compared to the corridor scenario. Similar to the values from table 1, the mean spacing for bottleneck is higher.

Scenario	mean
Bottleneck	1.895 ± 0.813
Corridor	1.656 ± 0.976

Table 3: Mean value of density

Estimating v_0, T, l In the next step our goal is to estimate the remaining unknown parameters of the Weidmann model. First we will start with estimating the speed under a free situation v_0. Once again the paper [10] does not specify more except that it was estimated using least squares fitting, which is a very vague statement. We propose the following methodologies to estimate the speed under a free situation v_0.

1. The first possible approach is to look into dataset at all instances where the pedestrian density D is low enough or mean-spacing $\overline{s}_K$ alows free movement. The main issue is that we do not know the exact values of D or $\overline{s}_K$ such that this is possible. Thus we can only assume that free movement is possible, when only one pedestrian is observed. To calculate this we would have to look at frames in the dataset where this condition holds. Note that this approach has the downside that we consider very few datapoints. Furthermore we might not be able to accurately calculate the speed v_0, because our observations are limited to the measurement area. Nonetheless this approach has the benefit as we are sticking close to the original dataset by calculating the average of the observed speed v_0.

2. The second approach is to use the original Weidmann model 3. The advantage of this approach is that we have γ and D_{max} given from [13]. Using these values we can calculate v_0 for each point and get a better estimation for v_0 by having access to more datapoints. But on the flipside this model is based originally on the Kladek model[7], which is a macroscopic speed model, since we are using the overall pedestrian density instead of the mean spacing of a specific pedestrian. using the calculated speed, we then used `scipy.optimize.curve_fitting(...)` to obtain the parameters T, l.

3. For the final approach we also used `scipy.optimize.curve_fitting(...)` as it is minimizing the square loss of the function 1. It is possible to obtain all 3 parameters v_0, T, l at once using this method. But since the results diverge quite a bit from the original, l remains as fixed parameter. We used the parameters T, l from the paper [10] from table 1. We then first try to obtain both v_0 and T and then perform curve fitting on only v_0.

1 For the first approach the results for approximating v_0 can be seen in table 4 (1). To obtain the approximations we filtered for frames where only 1 pedestrian is observed, hence for mean spacing being 0 or density being $1/(measurementarea)$. But overall these results can not be very accurate since we only have 54 datapoints for the bottleneck and 322 for the corridor scenario. Therefore they differ from the results in the paper as well but we do see the similarities as the speed in the bottleneck is higher than in the corridor case.

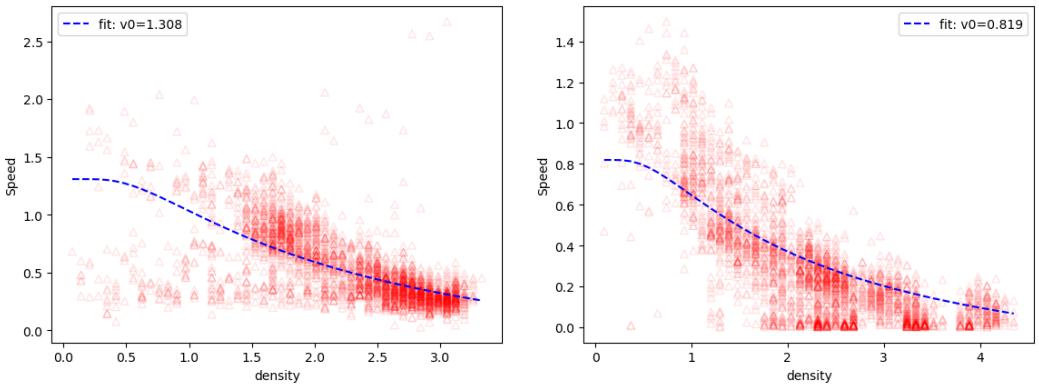
Scenario	v_0 (our)	v_0 (paper)
Bottleneck	1.009	1.64
Corridor	0.787	1.50

Table 4: v_0 estimated from data

The next problem is that we can't really estimate the values l and T from the data itself. In the paper l is defined as the physical size of a pedestrian. From literature on traffic flow [11] chapter 3.1 often length of a vehicle is referred to. In this context l would describe the length of a pedestrian in movement direction. The paper states here that l has the value $0.61m$ in the bottleneck scenario and $0.64m$ in the corridor scenario. Since this estimate is likely the result of measuring the length of the pedestrian from the top including their leg and feets, this value can not be reasonably be estimated. Therefore we set this value as fixed to $0.625m$.

For the time gap T the paper [10] describes it as the following time gap with the neighbors. It is not clear how exactly it is defined. We could assume that this time gap describes how long it takes for the neighbour that is closest behind one pedestrian to catch up with the one in front of him. T would then be the average of that between all pedestrians. This is rather difficult to directly compute from the dataset. We opted here to use curve fitting to obtain the missing parameters.

2 In the second approach we first obtained v_0 by curve fitting the original Weidmann model [13] in equation 3. Performing curve fitting on this model gives us the following figures 18. In table 5 we see that our bottleneck pedestrian speed in a free situation is higher now, but the one in the corridor has gone down. The speed for the bottleneck case, lines up with the speed of a pedestrian in a free situation mentioned in the literature by Weidmann [13] section 4.3.1, where $v_{F,hf} = 1.34$. For the corridor case it is surprisingly low, but it might be an artifact from having the different data preprocessing procedure.



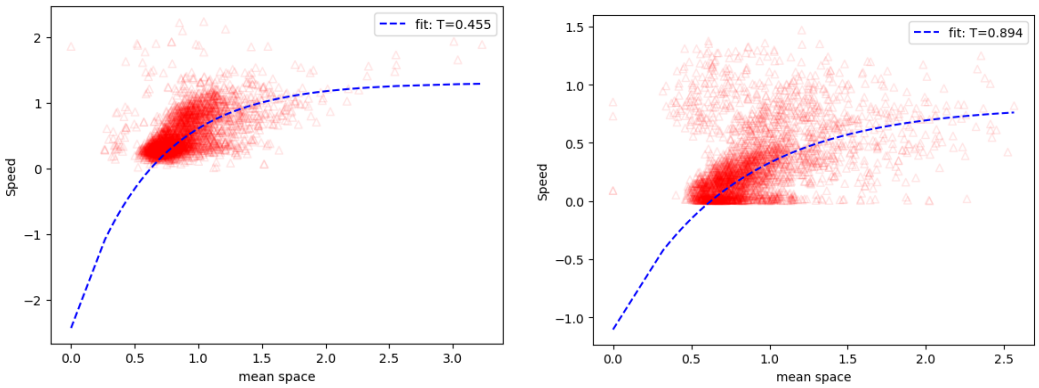
(a) Bottleneck (b) Corridor

Figure 18: Curve fitting v_0 on the densities

Scenario	v_0 (our)	v_0 (paper)
Bottleneck	1.308	1.64
Corridor	0.819	1.50

Table 5: v_0 from curve fitting density

To double-check if curve-fitting on the original Weidemann model was meaningful, we next perform curve-fitting once again using the obtained v_0. Additionally as previously mentioned we use the fixed physical size $l = 0.625$. Now we are left with obtaining the parameter T. In figure 19 we plotted the results after curve-fitting. This plot is similar to the one from the paper [10] figure 4. In table 6 we see that our results for the parameter T are quite similar to the one from the paper.



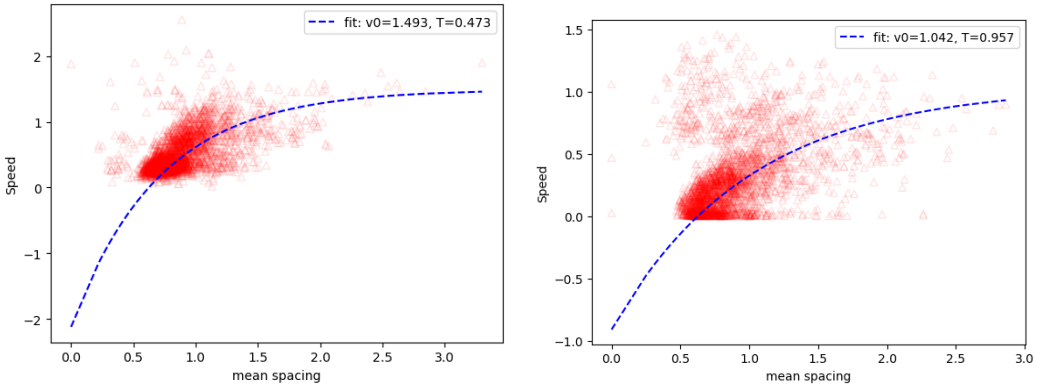
(a) Bottleneck (b) Corridor

Figure 19: Curve fitting T on original Weidmann function

Scenario	T (our)	T (paper)
Bottleneck	0.455	0.49
Corridor	0.894	0.85

Table 6: T from curve fitting original Weidmann function

3 Since curve-fitting can be performed on multiple parameters simultaneously, we will next try to obtain the parameter v_0 and T given $l = 0.625$. In figure 20 we see the resulting values for v_0 and T. In tables 7 and 9 we see that our results for speed are now closer to the ones in the paper. At the same time the results confirm that the corridor has a much lower speed for a pedestrian in a free situation. As for the parameter T in table 8 the values are still similar to the results from the paper, though for the corridor scenario T has a much higher value compared to the result obtained from the previous section.



(a) Bottleneck (b) Corridor

Figure 20: Curve fitting T on original Weidmann function

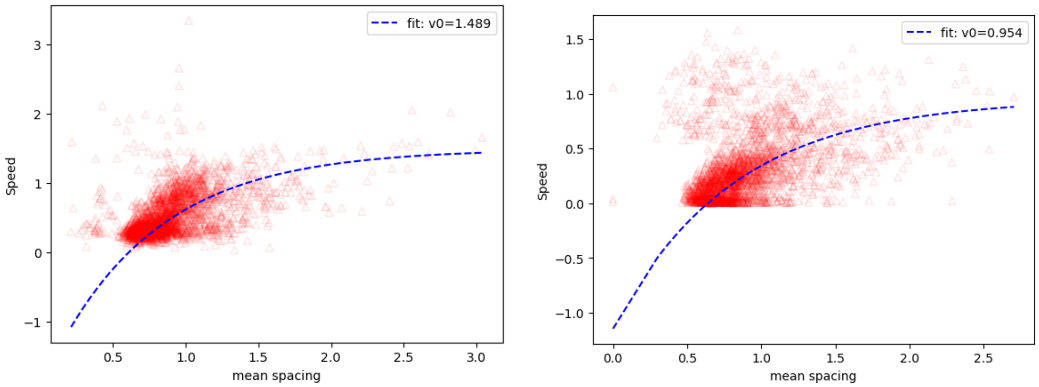
Scenario	v_0 (our)	v_0 (paper)
Bottleneck	1.493	1.64
Corridor	1.042	1.50

Table 7: v_0 from curve fitting equation 1

Scenario	T (our)	T (paper)
Bottleneck	0.473	0.49
Corridor	0.957	0.85

Table 8: T from curve fitting on equation 1

As our final test we are curve-fitting the parameter v_0 given T, l from the paper [10] in table 1. In figure 21 we can see similar results as seen from previous tests. Compared to the results from curve-fitting on 2 parameters here v_0 has a higher speed in the bottleneck scenario and lower speed the corridor case. Overall the results are similar to the tests performed previously.



(a) Bottleneck (b) Corridor

Figure 21: Curve fitting v_0 on original Weidmann function

Scenario	v_0 (our)	v_0 (paper)
Bottleneck	1.549	1.64
Corridor	0.923	1.50

Table 9: v_0 from curve fitting equation 1

Conclusion Overall we can obtain similar results for the parameter T. As for the parameter v_0 we can have similar results for the bottleneck scenario, but for the corridor scenario we have significantly lower values. This is likely due to different ways of preprocessing data, as it is not specified in the paper itself. But we could still say that relatively speaking, we can draw the a similar conclusion, since v_0 is higher in the bottleneck case compared to the corridor case.

In table 10 we see a final comparison the results from all approaches. Overall all results show that v_0 has a higher value for the Bottleneck scenario than the corridor scenario. In this case approach (1) is closer the results in the paper in terms of its ratio. We can conclude that our data processing approach is probably incomplete, as the pedestrian speed in a free situation v_0 shouldn't be too different between both scenarios.

Scenario	Bottleneck	Corridor	Ratio (B/C)
v_0 from paper	1.64	1.50	1.093
v_0 from (1)	1.009	0.787	1.282
v_0 from (2)	1.308	0.819	1.597
v_0 from (3.1)	1.493	1.042	1.433
v_0 from (3.2)	1.549	0.923	1.678

Table 10: v_0 from all approaches

Report on task 4, Comparison of the approaches (20%)

After implementing and testing the models we can now draw comparisons between both models. The main metric used to compare the neural network approach with the knowledge-based approach is the mean-squared error (MSE). Similar to the paper [10] we define this error as follows:

$$MSE(v, \hat{v}) = \frac{1}{n} \sum_{i=1}^{n} (v_i - \hat{v}_i)^2 \tag{4}$$

Here in 4 v is the velocity from the dataset (after preprocessing) and $\hat{v}$ is the predicted velocity using either approach. Apart from reproducing the results from [10] our goal is to see how well a neural network approach performs in comparison to a knowledge-based model.

Comparing results of both approaches First we have to note that for the knowledge-based approach the MSE was calculated separately for the bottleneck and corridor scenario. Since this approach fits slightly different parameters for each scenario, we will therefore not test a function fit for bottleneck scenario on data of the corridor scenario.

In the Table 11 we calculated the MSE for the methods on obtaining the parameters of the Weidmann model from the previous section. Most notably the MSE hovers around 0.117 for the bottleneck scenario and 0.103 for the corridor scenario. Taking this into context of the results for obtaining the parameters, we see that quite different parameters are possible to achieve similarly good results. Overall we do see that method 3.1 performs the best, since in this curve fitting approach both parameter T and l could be tweaked.

Scenario	Bottleneck	Corridor
$W(...)$ from (2)	0.11723	0.10359
$W(...)$ from (3.1)	0.11657	0.10294
$W(...)$ from (3.2)	0.11669	0.10342

Table 11: MSE for Weidmann model

In Table 12 we can observe the results for the experiment of Figure 12, one of the best ones for the neural network Task 2.2. Also we can see the result for the largest network in Task 2.1. For Task 2.2, we can see how the best network in the Bottleneck is the simplest one, the one with 3 neurons. However, the network with (10,4) is the one that achieves the best average result in both scenarios.

Scenario	Bottleneck	Corridor
(3) second order optimizer (Task 2.2)	0.076	0.099
(10,4) second order optimizer (Task 2.2)	0.075	0.054
(256, 128, 65, 32) (Task 2.1)	0.007	0.007

Table 12: MSE for a selection of neural networks in Task 2.1 and 2.2

In the comparison we can see that the neural network models perform better (far better in the case of Task 2.1 approach) than the Weidmann model, the knowledge-based model we used. As in [10], the neural network approach has less MSE than the Weidmann model for almost every experiment. This supports the authors' argument in the paper. While our results may not precisely match theirs, possibly due to variations in parameters or preprocessing techniques, the fundamental relationship between the performance of the knowledge-based model and the neural networks remains consistent.

Also, it is evident that the preprocessing in Task 2.1 is very useful as it allow us to use larger networks without overfitting, achieving results very close to the referenced paper. In contrast, it seems that the problem for this work is more a data problem than an architecture problem, as we can observe comparing results from Task 2.1 and Task 2.2.

It is interesting to note that despite employing different pretraining and parameter configurations, the optimal model from their NN_3 was also the one with 3 neurons in the hidden layer, mirroring our own findings for Task 2.2. This consistency is remarkable, especially considering that this simple network architecture yielded the best results for both studies. This observation was surprising in their paper, and it becomes even more

surprising for us since we are not using identical configurations. Therefore, it is plausible to conclude that the success of this network lies in its balance and the low data: it is sufficiently complex to perform well, yet not overly complex to overfit the limited amount of data available.

Benefits and Drawbacks After experimenting on both approaches we support the benefits and drawbacks suggested in paper [8].

For the DL approach, we do not need to find by ourselves a set of rules or equations that is congruent with the dynamics of the pedestrians in all scenarios. Relying on the big amount of data we have about crowd dynamics, we do not need to have a profound domain knowledge to actually getting very good predictions. Also, theoretically, the neural networks are able to learn really complex patterns if we have enough amount of data. On the other hand, the dynamics that the neural network is going to learn are not explicable and their parameters are non-interpretable. Depending on the context, this can be a serious drawback. For example, if you are using a model to predict crowd dynamics for a security protocol in a building, it is safer and ethical for the model to be explainable. In case of an accident caused by a wrong prediction, it is important to be able to examine the model to understand why it made the mistake.

The knowledge-based model faces it biggest challenge in finding the methodology to describe the model and obtain the parameters of the model. Additionally data preparation is time-consuming, though this applies to any problem dealing with large amounts of data. Overall the main benefit of the knowledge-based approach lies in its efficiency and speed to obtain model parameters that achieve very low errors. While the errors could be lower, this approach tends to use a rather simple function, which also makes the model very easy to interpret. Overall knowledge-based models are best used when the problem is describable as a function, since this function can then be easily fitted to the data, because only few parameters need to be tuned. But on the flipside, their accuracy are not near-optimal, as more complex non-linear functions should always be able to achieve better results.

Challenges of the KB approach Overall the implementation of the Weidmann model is extremely simple and performing tests to measure its MSE can be done quite easily. The main issue of fitting the Weidmann model to the original data lies in the following 2 points.

1. Preprocessing the data
2. Obtaining the parameters of the Weidmann model

The first issue is preprocessing the data. Apart from us not knowing the exact method that was used to preprocess the data that was used for [10], preprocessing itself has many steps and things to consider. Preparing the data depends first off on how the data is going to be used. For the Weidmann model the idea is to fit the Weidmann model on a diverse set of scenarios. So all data from each scenario are used to obtain parameters separate for each scenario, those being Bottleneck and Corridor scenario. Some considerations that had to be made additionally are deleting z-positions as they are not used. Also positions that outside the measurement area had to be removed, though here an arguemnt can be made about how these positions could be used to calculate the speed of each pedestrian.

After basic cleaning additional data needs to be generated, which is later needed for fitting the Weidmann model onto the data. The Weidmann model mainly uses the mean spacing as input to calculate the speed of a pedestrian. Therefore mean spacing and speed needs to be calculated for each pedestrian at each frame. Again the exact methodology of the authors from [10] is unknown and we can only speculate how border cases are handled. One example being if there are less than 10 neighbours, whether we should calculate a mean spacing with the remaining neighbours or immediately set it to a default value. Also one needs to consider if pedestrian outside of the measurement area affect the mean spacing. Overall data preprocessing presents one of the biggest issues when dealing with a knowledge-based model as the results of fitting the parameters are directly tied to the quality of your data.

As for fitting the model we once again encountered the same issue as the authors have only vaguely described how the parameters for the Weidmann model were obtained. As we can see from our previous section on the knowledge-based approach, we have tried different ways to obtain the parameters v_0, T, l for the Weidmann model. The paper merely mentions that it used least-squares fitting to obtain those models. The closest approach to this description was using curve fitting with least-squares from the `scipy.optimize` library. Even here there are different ways how the curve fitting approach can be applied. Additionally the knowledge-based model lacks immediate interpretability, when descriptions are missing. The paper [10] describes very poorly

what the parameters T and l exactly represent and how they were obtained. From studying other literature [11] we could find out that l commonly describes the length of a vehicle in the context of traffic flow analysis. Applying to the context of pedestrians l hence describes the length of a pedestrian in their walking direction when looking from above. Another side note, is that we couldn't find were the Weidmann model mentioned in the paper [10] was actually derived from. While it cites an older document from Weidmann [13], this paper only mentions a similar model, which puts the pedestrian density into relation with the speed, instead of mean spacing with the speed. It was thus confusing to work with the Weidmann model, as there was barely any literature on it. Oour only directions to a source led to a different model, because the Weidmann model[10] is a microscopic model, while the Weidmann model[13] is a macroscopic model and also was originally from the Kladek formula [7] and we had no access to this old document from 1966.

Challenges of the NN approach As we have already commented in the challenges for the KB approach, also in the neural network approach the main difficulties are the preprocessing of the data and hyperparameter tuning. The tuning typically demands a exploration and extensive runtime for learning, necessary to obtain accurate estimates of hyperparameter quality. Furthermore, selecting the appropriate features and determining the optimal loss function added more difficulties. We think the paper should provide clearer and more detailed explanations of the experimentation setup so that others can easily reproduce their results.

Report on task 5, Discussion of future work and other approaches (5%)

The paper [8] talks in the discussion and future work session about new trends in the approximation of pedestrian dynamics.

Hybrid approaches They mention the hybrid approaches that mix neural networks and knowledge-based approaches as a very valuable alternative, as many drawbacks of DL algorithms are strengths in the knowledge-based approach, like the interpretability of its parameters. They describe two ways of combining the two concepts:

- **Use Knowledge-based models to improve DL algorithms** Some interesting examples of these are:
  - **Data Generation:** Knowledge-based models generate synthetic data sets to augment training data for the DL algorithm, particularly useful when real-world data is limited or lacks diversity. For instance, simulations based on KB models can enhance training data for scenarios not well-represented in existing data sets [12].
  - **Knowledge-Guided Design of Architecture:** Knowledge-based is integrated into the architecture design of DL models to capture domain-specific dependencies among variables. This involves embedding knowledge directly into the network structure [1].
  - **Knowledge-Guided Loss Function:** Knowledge is incorporated into the loss function of DL algorithms to ensure outputs adhere to physical laws, minimizing unrealistic predictions [14].
- **Use DL algorithms to improve the prediction of knowledge-based models:** This can be done for example by training to correct errors made by KB models by predicting model residuals, as in [14] too. Also, DL algorithms can be used to adjust the parameters of KB models, as they did to refine the social force model parameters in [3].

As we can see the authors conclude that neural network and knowledge-based approaches can be used as complementary methods to strengthen each other's weaknesses.

As we saw in Task 2, we tried an hybrid approach as it seemed like an interesting way to try to improve our results with the neural net. Following the different types of hybrid models suggested by the paper, ours is closer to the former, and within it, closer to the *knowledge-guided design of architecture* as we are changing the last layer to resemble the Weidmann model. However, the hybrid approach we designed was not that better compared to the rest, being a little bit better than other models we tried. This could happen also because our hybrid model is very simple and rudimentary, nevertheless we think that, in general, hybrid models would be refined in the future leading to better results, as they have "the better of both worlds".

On the other hand, in [8] they also discussed other directions for the future. For example, Reinforcement learning algorithms are being used, as they sometimes resemble how people learn and move. Reinforcement learning is a common method also to made agents learn about its environment by setting a reward for reaching a goal while avoiding collisions. However, RL methods need a reward function, that sometimes is not easy to model, and a priori goal or destination. There are some works where they combine RL and DL approaches to try to mitigate these drawbacks [2].

Finally, the propose some questions regarding the possibility of a successful use of DL algorithms in large-scale crowd simulations. In our opinion, as we have observed in our report and experience with the previous tasks, we think that one of the main problems of DL is that it needs a lot of data. Moreover, we have seen that it is difficult for them to learn more insightful patterns of the pedestrian dynamics. Specifically, when trained on a specific scenario, they tend to learn the dynamics unique to that scenario rather than general patterns of movement. Consequently, when tested in a new scenario, performance tends to degrade. This limitation in generalization in new environments is common to neural networks across various domains, but it holds particular significance in crowd simulation due to the multitude of potential scenarios. We hope that in the future, advancements in deep learning will address this limitation, making it even more beneficial for crowd simulation tasks.

References

[1] Alessandro Antonucci, Gastone Pietro Rosati Papini, Luigi Palopoli, and Daniele Fontanelli. Generating reliable and efficient predictions of human motion: A promising encounter between physics and neural networks. *arXiv preprint arXiv:2006.08429*, 2020.

[2] Michael Everett, Yu Fan Chen, and Jonathan P How. Collision avoidance in pedestrian-rich environments with deep reinforcement learning. *IEEE Access*, 9:10357–10377, 2021.

[3] Simone Göttlich and Stephan Knapp. Artificial neural networks for the estimation of pedestrian interaction forces. *Crowd Dynamics, Volume 2: Theory, Models, and Applications*, pages 11–32, 2020.

[4] Leroy F Henderson. On the fluid mechanics of human crowd motion. *Transportation research*, 8(6):509–515, 1974.

[5] LF Henderson. The statistics of crowd fluids. *nature*, 229(5284):381–383, 1971.

[6] C Keip and K Ries. "dokumentation von versuchen zur personenstromdynamik". *Project Hermes, Bergische Universität Wuppertal, Tech. Rep*, 2009.

[7] Horst Kladek. *Über die Geschwindigkeitscharakteristik auf Stadtstraßenabschnitten.* PhD thesis, 1966.

[8] Raphael Korbmacher and Antoine Tordeux. Review of pedestrian trajectory prediction methods: Comparing deep learning and knowledge-based approaches. *IEEE Transactions on Intelligent Transportation Systems*, 2022.

[9] James Martens and Roger Grosse. Optimizing neural networks with kronecker-factored approximate curvature, 2020.

[10] Antoine Tordeux, Mohcine Chraibi, Armin Seyfried, and Andreas Schadschneider. Prediction of pedestrian dynamics in complex architectures with artificial neural networks. *Journal of intelligent transportation systems*, 24(6):556–568, 2020.

[11] Martin Treiber and Arne Kesting. Traffic flow dynamics. *Traffic Flow Dynamics: Data, Models and Simulation, Springer-Verlag Berlin Heidelberg*, pages 983–1000, 2013.

[12] Laura von Rueden, Sebastian Mayer, Rafet Sifa, Christian Bauckhage, and Jochen Garcke. Combining machine learning and simulation to a hybrid modelling approach: Current and future directions. In *Advances in Intelligent Data Analysis XVIII: 18th International Symposium on Intelligent Data Analysis, IDA 2020, Konstanz, Germany, April 27–29, 2020, Proceedings 18*, pages 548–560. Springer, 2020.

[13] Ulrich Weidmann. Transporttechnik der fußgänger: transporttechnische eigenschaften des fußgängerverkehrs, literaturauswertung. *IVT Schriftenreihe*, 90, 1993.

[14] Jared Willard, Xiaowei Jia, Shaoming Xu, Michael Steinbach, and Vipin Kumar. Integrating scientific knowledge with machine learning for engineering and environmental systems. *ACM Computing Surveys*, 55(4):1–37, 2022.