Modeling rush hour vehicular traffic using a machine learning approach

Erik Ackzell

Master's thesis
2022:E76

Lund University

Faculty of Science
Centre for Mathematical Sciences
Numerical Analysis

Centrum Scientiarum Mathematicarum

Modeling rush hour vehicular traffic using a machine learning approach

Erik Ackzell

supervised by Alexandros Sopasakis

November 27, 2022

Abstract In this thesis, a convolutional neural network is used to model the behaviour of individual vehicles on a stretch of the U.S. 101 highway during rush hour. This model is then extended to model the collective behaviour of all vehicles on the stretch of road and a 15 minute simulation is carried out. Using an initial vehicle layout, images displaying the position of nearby vehicles and information of vehicles entering the road, the simulation performs well for the first five minutes, while the performance deteriorates for the subsequent 10 minutes.

Populärvetenskaplig sammanfattning på svenska

Det finns olika typer av så kallade artificiella neurala nätverk som används till olika saker. En av dessa kallas för konvolutionella neurala nätverk och används till att bearbeta bilder. Denna typ av artificiella neurala nätverk används bland annat för klassificering av bilder och för att styra självkörande bilar.

I den här uppsatsen används ett konvolutionellt neuralt nätverk för att försöka förutspå hur en bil ska bete sig, baserat på en bild uppifrån som visar var andra bilar befinner sig i närheten av den aktuella bilen. Sedan försöks det förutspås hur alla bilar på en del av en motorväg i USA beter sig i rusningstrafik. Även om det går att delvis förutspå beteendet hos bilarna, upptäcks det att det inte räcker med enbart en bild uppifrån utan att det krävs ytterligare information.

For Alvar and Anna

Contents

1 Introduction **5**
- 1.1 Aim and research questions . . . 5
- 1.2 Limitations . . . 5

2 Theory **6**
- 2.1 Artificial neural networks . . . 6
  - 2.1.1 Nodes and layers . . . 6
  - 2.1.2 Weights and biases . . . 7
  - 2.1.3 Activation functions . . . 7
  - 2.1.4 Training . . . 9
- 2.2 Gradient descent . . . 10
- 2.3 Different types of artificial neural networks . . . 13
  - 2.3.1 Fully connected feedforward . . . 13
  - 2.3.2 Recurrent . . . 13
  - 2.3.3 Convolutional . . . 13

3 Method **14**
- 3.1 Original data and stochastically generated data . . . 14
- 3.2 Determining hyper parameters for CNN . . . 18
  - 3.2.1 Using stochastically generated images . . . 18
  - 3.2.2 Determining hyper parameters . . . 19
- 3.3 Building data set from real data . . . 21
  - 3.3.1 Creating images . . . 21
  - 3.3.2 Creating database . . . 22
- 3.4 Train final CNN . . . 24
- 3.5 Extend CNN . . . 24

4 Results **25**
- 4.1 Determining hyper parameters . . . 25
- 4.2 Final CNN performance on individual vehicles . . . 27
- 4.3 Model performance compared to reality . . . 27

5 Discussion and conclusion **29**
- 5.1 Discussion of results . . . 29
- 5.2 Possible improvements to method . . . 30

1 Introduction

In this project, we model the collective behavior of vehicles on a stretch of the U.S. 101 highway in Los Angeles. We use a convolutional artificial neural network (CNN) to model the behavior of each individual vehicle and the model is then extended to describe the interaction of all the vehicles on the road. Apart from an initial state, the inputs to the model are the timestamps and lane numbers of the vehicles entering the road, and a simulation of approximately 15 minutes is carried out.
After the introduction, in which the aim, research questions and limitations are presented, the mathematical theory of the methods used in this project is given in Section 2. After this the different phases of the work is described in Section 3, which takes us to Section 4 in which the results are presented. Lastly, the results are discussed and possible improvements to the method are presented in Section 5.

1.1 Aim and research questions

The overall aim of the thesis is to

- Create a model based on an artificial neural network (ANN) using visual data to simulate rush hour traffic on a stretch of the U.S. 101 highway in Los Angeles.

This is achieved with the help of the following research questions:

- What type of ANN is suitable for modeling a single vehicle's behavior based on visual data?
- How can such an ANN be extended to a model which describes all vehicles on the stretch of road?
- How well does such an extended model perform when compared to reality in terms of modeling flow and velocity of all the vehicles on the stretch of road?

1.2 Limitations

The scope of the project has the following limitations:

- Only ANN:s which can be implemented in the Python library Keras are considered
- Only ANN:s which use a single type of activation function for all nodes in a network are considered

2 Theory

In this section, the mathematical frameworks of the methods used in this project are described, starting with a general introduction to feedforward artificial neural networks in subsection 2.1, after which the gradient descent method and variations of it are discussed in subsection 2.2. Lastly, different types of artificial neural networks are presented in subsection 2.3.

2.1 Artificial neural networks

Let $m, n, o \in \mathbb{N}$, $X \subset \mathbb{R}^{m \times n}$, $Y \subset \mathbb{R}^o$ and

$$f : X \to Y. \tag{1}$$

Given a sample of inputs

$$X_s \subset X \tag{2}$$

and a corresponding sample of outputs

$$f(X_s) = Y_s \subset Y, \tag{3}$$

one might want to determine an approximation of f, say f_{nn}. One possible candidate for such an approximation is an artificial neural network (ANN).
As the name suggests, the structure of an ANN is somewhat similar to that of a biological neural network [Sil+17] and consists of a number of *layers* of *nodes* (section 2.1.1) with information being passed between the nodes. Many comparisons between ANN:s and biological neural networks have been presented, see e.g. [Sha16], [Sil+17] and [DS14].
In a *feedforward* neural network (FNN), the flow of information between the nodes is one-directional, while other types of ANN:s, e.g. variants of *recurrent neural networks* (RNN) or two-dimensional lattice networks, allow for information to flow in more than one direction [DS14]. In an FNN, each of the nodes in a layer processes inputs from nodes in the previous layer and then passes the processed information to the nodes in the subsequent layer. These nodes can process the input data in different ways, depending on the choice of *activation function* (section 2.1.3) used. Each connection between two nodes has an associated *weight* while each node has an associated *bias* (section 2.1.2). In order to find a good candidate for f_{nn}, a *training* step is carried out (section 2.1.4).

2.1.1 Nodes and layers

An FNN consists of two or more *layers* containing *nodes*. The layers are ordered and are made up of

(i) one input layer,

(ii) n hidden layers, $n \in \mathbb{N}$, and

(iii) one output layer.

Let X, Y be as in (1). The input layer takes an element $x \in X$ that is sequentially transformed in the hidden layers and the output layer returns an element $y \in Y$. As input to each node, a linear combination of the outputs from the previous layer of nodes is used.

2.1.2 Weights and biases

The coefficients of the terms in the linear combinations used as input to each node in the hidden and output layers are called the *weights* $w_{qr}^{(s)}$ and *biases* $b_r^{(s)}$ of the ANN.
Let $y_i^{(j)}$ be the output of node number i in layer j and assume that layer j has exactly N_j nodes. Then the input to node k in layer $j+1$ is given by

$$x_k^{(j+1)} = \sum_{i=1}^{N_j} w_{ik}^{(j)} y_i^{(j)} + b_k^{(j)} \tag{4}$$

2.1.3 Activation functions

In order for the ANN to be able to approximate non-linear functions, the linear combination (4) is transformed using a non-linear function σ called an *activation function* such that the output of node k in layer $j+1$ is given by $\sigma(x_k^{(j+1)})$. In order to mimic the behavior of biological neural networks, with neurons either firing or not [Sha16], a naive first choice of an activation function is

$$\begin{aligned} &\hat{\sigma} : \mathbb{R} \to \{a, b\} \subset \mathbb{R}. \\ &\hat{\sigma}(x) = \begin{cases} a & x < c \\ b & x \geq c \end{cases}, \end{aligned} \tag{5}$$

for some $c \in \mathbb{R}$. However, choosing such an activation function has drawbacks that will be discussed in Section 2.2. One issue with such a function is that its derivative is always zero. Thus, other functions which have similarities to $\hat{\sigma}$ but whose derivatives are not constantly zero are often used. A selection of such functions is shown in Figure 1. If the activation functions would all be linear, the FNN would be a sequence of linear transformations of the input, which would collapse to a linear function. The activation function used in the output layer should have a codomain corresponding to the possible outputs of the function that is approximated. However, the activation functions to use in the hidden layers can be chosen regardless of the outputs. There are a large number of possible activation functions available in the Python Keras library [22] (see section 3) and what choice of activation functions that are going to result in the best performing ANN is data dependent [DSC21]. Below, a selection of activation functions is described.

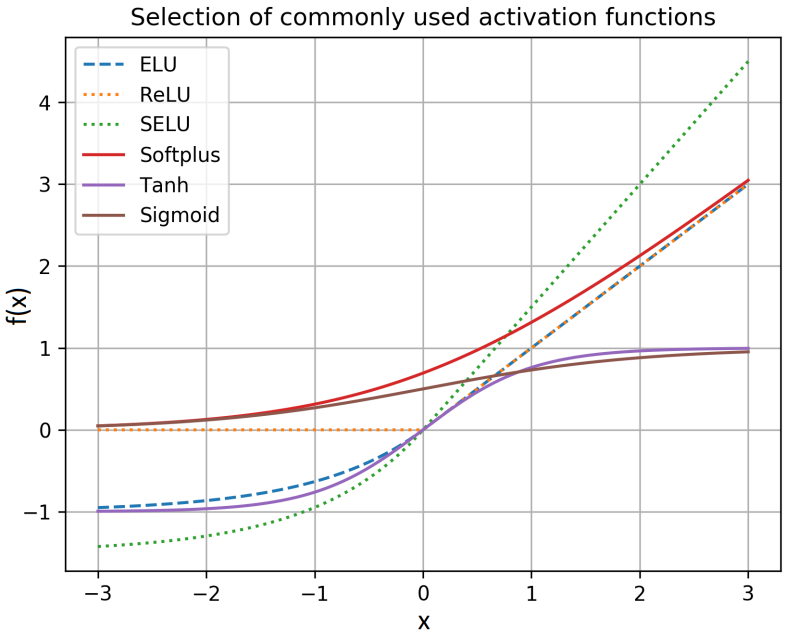

Figure 1: Some commonly used activation functions.

Exponential linear unit (ELU) [CUH15] is defined by

$$f_\alpha(x) = \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{6}$$

$$\mathrm{Im}(f_\alpha) = (-\alpha, \infty)$$

Scaled exponential linear unit (SELU) [Kla+17] is defined by

$$f_{\lambda,\alpha}(x) = \lambda \begin{cases} x & x > 0 \\ \alpha(e^x - 1) & x \leq 0 \end{cases} \tag{7}$$

$$\mathrm{Im}(f_{\lambda,\alpha}) = (-\lambda\alpha, \infty)$$

Rectified linear unit (ReLU) [GBB11] is defined by

$$f(x) = \max(x, 0) = x_+ \tag{8}$$

$$\mathrm{Im}(f) = (0, \infty)$$

Softplus activation function [GBB11] is given by

$$f(x) = \ln(e^x + 1), \tag{9}$$
$$\mathrm{Im}(f) = (0, \infty)$$

where ln is the natural logarithm.

Sigmoid activation function is given by

$$f(x) = \frac{1}{e^{-x} + 1} \tag{10}$$
$$\mathrm{Im}(f) = (0, 1)$$

2.1.4 Training

In order to determine a good candidate for f_{nn}, a *training step* is performed. In this step, an appropriate cost function is chosen, which is then minimized by some optimization method. Given a specific layout of the ANN in terms of number of layers, number of nodes in each layer and what activation functions to use, the parameters that make the ANN unique are the chosen weights and biases. The training is an iterative optimization process, in which the goal is to determine the weights and biases minimizing some *cost function*. These training iterations are called *epochs*.

Cost functions Let f, X, Y, X_s and Y_s be as in (1)-(3). Assume that and we want to determine the weights and biases of an ANN with $n > 3$ layers and that layer i has n_i nodes, yielding the space

$$W \subset \mathbb{R}^M, \qquad M = \sum_{i=2}^{n} n_i(n_{i-1} + 1) \tag{11}$$

of weights and biases. Let $\mathbf{w} \in W$ and denote this ANN by $f_{\mathbf{w}}$. We can then define a cost function

$$J : \mathbb{R}^M \to \mathbb{R}_+, \tag{12}$$

measuring the difference between $Y_s = f(X_s)$ and $\hat{Y}_s = f_{\mathbf{w}}(X_s)$.
Let N be the number of elements in X_s and x_i, $i = 1, 2, \ldots N$ the elements of X_s.
Common choices of cost functions are the mean squared error

$$J_{MSE}(\mathbf{w}) = \frac{\|Y_s - \hat{Y}_s\|_2^2}{N}, \tag{13}$$

and the mean absolute error

$$J_{MAE}(\mathbf{w}) = \frac{\|Y_s - \hat{Y}_s\|_1}{N}. \tag{14}$$

Over fitting A potential risk when minimizing the cost function is that the found weights and biases $\mathbf{w}$ causes the ANN to very accurately map the elements in X_s to the corresponding elements in Y_s but fail to do so for elements in $X - X_s$ [Sil+17]. We call this phenomenon *over fitting*. There are a number of ways to handle combat over fitting. One is to introduce *dropout* [Sch16]. During each iteration of the training, a random selection of nodes is then deactivated, effectively training a large set of different ANN:s and then combining them all. Another way is to introduce early stopping [Sil+17]. This is done by randomly dividing the available data in two parts, a *training data set* and a *hold out data set*. The training data set is used to train the network and after each epoch, the current version of the network is used on the hold out data set. When cost function starts to increase for the hold out data set from one epoch to the next, the training is stopped. This has the effect of preventing the network from adapting to the noise in the training data and generalise better to new unseen data.

2.2 Gradient descent

Let $f : \mathbb{R}^n \to \mathbb{R}$ be a smooth function. The *gradient descent* or *delta rule* method is an iterative method used to minimize such a function. It makes use of that the gradient of f, evaluated at a point $w^{(0)} \in \mathbb{R}^n$, points in the direction of the steepest ascent of f. Thus, setting

$$w^{(1)} = w^{(0)} - \alpha \nabla f(w^{(0)}) \tag{15}$$

for some appropriate choice of the *learning rate* $\alpha \in \mathbb{R}^+$, we achieve

$$f(w^{(1)}) < f(w^{(0)}). \tag{16}$$

If f is convex and α is appropriately updated, we will eventually find a $w^{(N)}$ such that

$$f(w^{(N)}) < f(w) \quad \forall w \in \mathbb{R}^n, \tag{17}$$

for some $N \in \mathbb{N}$. Gradient descent is a method which is often used to minimize the cost functions associated with the training of ANN:s. One of the reasons to not use higher order methods, e.g. the second order Newton's method, is that the number of variables associated with the these cost functions can be extremely large, making second derivative calculations very costly.
There is a multitude of variations of the gradient descent method, which are being used in the context of ANN:s. The variations can each be put into one of two categories, which propose different methods of

(i) handling the large number of training examples and

(ii) performing the updates of the weights and biases,

respectively.

Ways to handle the large number of training examples These methods reduces computation time by selecting only parts of the training examples to be evaluated during each epoch [Rud16]. As a reference point, we first describe the batch gradient descent method, in which all training examples are used in each epoch.

Batch gradient descent In the batch gradient descent method, the output of the ANN is calculated for all training examples, and all of the errors are then combined in the cost function. When the number of training examples are in the millions or billions, this could be quite time consuming.

Stochastic gradient descent (SGD) In this method, instead of including the output of the ANN of all the training examples before calculating the cost function and subsequently updating the weights and biases, this is instead done after each single training example.

Mini-batch gradient descent A combination of batch gradient descent and SGD is the mini-batch gradient descent, in which the set of training examples is divided into smaller batches, for which the cost function and its gradient is calculated and the weights and biases are updated.

Ways to perform the update step In the standard gradient descent method, the learning rate as can be seen in equation (15), is set beforehand and is not updated throughout the execution of the algorithm. This is something that some of the following methods address [Rud16].

Momentum In the momentum method, ∇f in equation (15) is replaced by a different vector $d^{(0)}$. In the first iteration of the algorithm, $d^{(0)} = \nabla f(w^{(0)})$. However, in the following iterations the update looks as follows

$$d^{(i)} = \beta d^{(i-1)} + \alpha \nabla f(w^{(i)}) \tag{18a}$$

$$w^{(i+1)} = w^{(i)} - d^{(i)}, \tag{18b}$$

where $\beta \in \mathbb{R}$. In other words, we include a multiple of the update vector from the previous iteration when performing the current update. A geometric motivation behind this is to speed up the performance of the algorithm in case the current value $w^{(i)}$ is mapped to the side of a valley of the graph of the cost function, while the valley itself would slope less towards its bottom. In such a case, convergence could become slow, as each update could produce a value on the opposite side of the valley, only slightly progressing towards the actual bottom of the valley, since the main component of the gradient would point towards the other side.

Nesterov accelerated gradient (NAG) In NAG, the idea is to refine update rule in equation (18) by taking the first term $\beta d^{(i-1)}$ in equation (18a) into consideration before calculating the gradient. Instead of evaluating the gradient at $w^{(i)}$, we evaluate it at $w^{(i)} - \beta d^{(i-1)}$. That is, we replace equation (18) by the following update rule

$$d^{(i)} = \beta d^{(i-1)} + \alpha \nabla f(w^{(i)} - \beta d^{(i-1)}) \tag{19a}$$

$$w^{(i+1)} = w^{(i)} - d^{(i)}. \tag{19b}$$

The motivation behind this is that $w^{(i)} - \beta d^{(i-1)}$ is a better approximation of $w^{(i+1)}$, and so evaluating the gradient at this point better corresponds to the behavior of the function at the next time step.

Adagrad Adagrad is a method which, instead of using a single, fixed learning rate, uses individual learning rates for different parameters and adapts these over time. Given some initial general learning rate α, we use the update rule

$$w_k^{(i+1)} = w_k^{(i)} - \frac{\alpha}{\sqrt{\sum_{j=0}^{i} \left(\frac{d}{dw_k} f(w^{(j)}) \right)^2 + \epsilon}} \cdot \frac{d}{dw_k} f(w^{(i)}). \tag{20}$$

In other words, the learning rate is altered through division of the sum of the squares of the derivative w.r.t. the corresponding parameter for all previous time steps. The term ϵ is included to avoid division by zero.
Equation (20) implies that for a specific parameter w_k, the relative relation

$$\frac{d}{dw_k} f(w^{(j)}) \text{ large for previous time steps } \Rightarrow \text{smaller learning rate} \tag{21}$$

holds.

Adadelta Adadelta is a refinement of Adagrad. Here, the sum over all the squared derivatives in the denominator in equation (20) is replaced in order to allow the more recent derivatives to have a higher significance and to avoid the possibility of a very large denominator. The sum is replaced by a combination of the squared latest derivative and an average of the previous derivatives. We define the denominator D as

$$D = \sqrt{\frac{\lambda}{i} \sum_{j=0}^{i-1} \left(\frac{d}{dw_k} f(w^{(j)}) \right)^2 + (1-\lambda) \left(\frac{d}{dw_k} f(w^{(i)}) \right)^2 + \epsilon}, \tag{22}$$

which yields the expression

$$w_k^{(i+1)} = w_k^{(i)} - \frac{\alpha}{D} \cdot \frac{d}{dw_k} f(w^{(i)}). \tag{23}$$

Lastly, we replace the learning rate α in (23) with a combination of the squared latest weight update d_{i-1} and previous weight updates, resulting in the nominator N

$$N = \sqrt{\frac{\lambda}{i-1}\sum_{j=0}^{i-2} d_j^2 + (1-\lambda)d_{i-1}^2 + \epsilon} \tag{24}$$

RMSprop RMSprop is another refinement of Adagrad very similar to Adadelta. It is defined by equations (22) and (23) with $\lambda = 0.9$.

2.3 Different types of artificial neural networks

There are different classes of ANN:s for different use cases. In this section, three such classes are described.

2.3.1 Fully connected feedforward

In the *fully connected feedforward* network, the information flows in one direction only and all nodes in one layer is connected to all nodes in the previous and following layer [DS14].

2.3.2 Recurrent

In the fully connected feedforward network, input data from one epoch is not retained in subsequent epochs. In order to be able to extract more information from time series data, the *recurrent* network was introduced [Ben91].

2.3.3 Convolutional

The *convolutional* artificial neural network (CNN) is a feedforward network, but has at least one convolutional layer which uses the *convolution operator*. This type of network is used to process image input as tensors. In order to perform the convolution operation, we define a number of *kernels* [ON15]. Let $T \in \mathbb{R}^{o\times p\times q}$ be an input tensor and $k \in \mathbb{R}^{m\times n}$ such a kernel. We define the convolution operator, here denoted by $\odot$, such that each element in the resulting tensor $S = k \odot T \in \mathbb{R}^{(o-m+1)\times(p-n+1)\times q}$ is given by

$$S_{a,b,c} = \sum_{i=1}^{m}\sum_{j=1}^{n} k_{i,j}T_{(i+a-1),(j+b-1),c} \qquad c = 1, 2, \ldots, q \tag{25}$$

An activation function σ is then applied to each of the elements of S in order to introduce non-linearity as described in Section 2.1.3. Each kernel results in one *output channel* and after one or more convolutional layers, a *pooling* layer or a *flatten* layer is used to reduce the order of the output for use in subsequent layers. Denote this flatten operator by F and let $Q \in \mathbb{R}^{\alpha\times\beta\times\gamma}$ be an output tensor from one channel of the last convolutional layer of the network. Then $F(Q)$ contains the same information as Q but restructured such that $F(Q) \in \mathbb{R}^{\alpha\cdot\beta\cdot\gamma}$.

3 Method

In this section the process of the project is described. In section 3.1, the original data used is described, in 3.2 we describe how the hyper parameters used for the final CNN were determined, in 3.4 we describe how the final CNN is trained and lastly how the CNN is then extended to predict the behavior of all vehicles on the road section in question.

As the computations needed in the project are impossible to carry out by hand, programs written in the Python language are used. In the project, the collection of implementations of numerical methods used for ANN:s called Keras is used.

3.1 Original data and stochastically generated data

The original real-world data used consists of measurements of vehicles on a 2100 ft stretch of US highway 101 made by Cambridge Systematics Inc. for the US Federal Highway Administration on June 15, 2005 between 07.50 and 08.05. Data was recorded with a frequency of 10Hz. See Table 1 for a description of the data set and Figure 2 for an image of the road section in question.

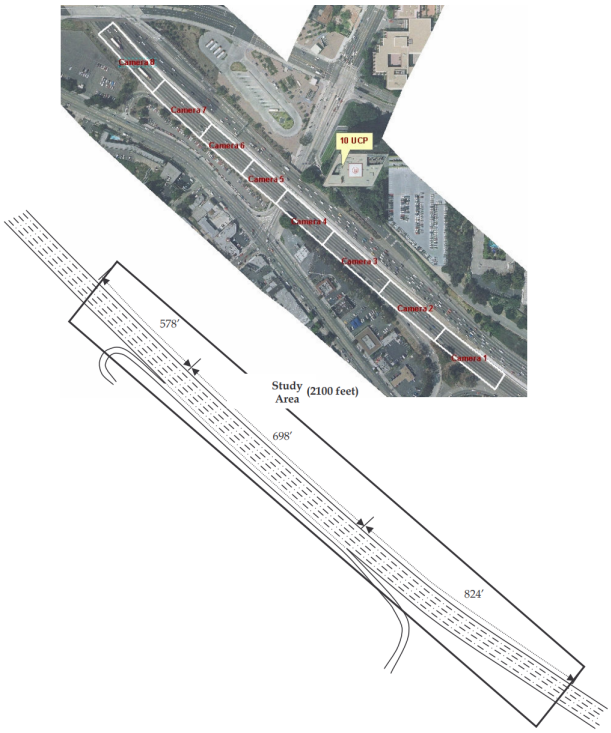

Figure 2: The road section at which measurements were taken.

Column	Description	Details
Vehicle ID	Vehicle identification number	Each vehicle is identified by a unique integer.
Frame ID	Frame identification number	Each time point is identified by a unique integer representing the number of deciseconds from the time when the data collection began.
Total frames	Total number of frames for the vehicle	The total number of time points at which a measurement of the particular vehicle is recorded.
Global time	Current time	Current time measured in milliseconds since Jan 1st, 1970.
Local X	Local lateral coordinate	Number of feet from the left-most edge of the road section.
Local Y	Local longitudinal coordinate	Number of feet from the entry level of the road section.
Global X	Global lateral coordinate	Lateral coordinate according to CA State Plane III.
Global Y	Global longitudinal coordinate	Longitudinal coordinate according to CA State Plane III.
Vehicle length	Length of vehicle	Length of vehicle measured in feet
Vehicle width	Width of vehicle	Width of vehicle measured in feet.
Vehicle class	Type of vehicle	1 - motorcycle, 2 - auto, 3 - truck.
Vehicle velocity	Current velocity of vehicle	Current velocity of vehicle measured in feet per second.
Vehicle acceleration	Current acceleration of vehicle	Current acceleration of vehicle measured in feet per second square.
Lane identification	Current lane of vehicle	1 - farthest left, 5 - farthest right, 6 - auxiliary lane between on-ramp and off-ramp, 7 on-ramp, 8 off-ramp.
Preceding vehicle	Vehicle ID of preceding vehicle	0 - no preceding vehicle.
Following vehicle	Vehicle ID of following vehicle	0 - no following vehicle.
Spacing	Space headway	Distance in feet between the front-center of vehicle to the front-center of the preceding vehicle.
Headway	Time headway	Time in seconds to travel at present speed from the front-center of the vehicle to the front-center of the preceding vehicle.

Table 1: Description of original data set

The data contains information about the complete trajectories of 2169 vehicles, with an average length of 14.77 ft.

Apart from the real-world data the project uses stochastically generated data consisting of a data set and corresponding images based on the real-world data. See Table 3 for a description of the data set. Some measurements describing the data set can be seen in Table 2 and a histogram showing the distribution of the steering column can be seen in Figure 3. The images consist of triples, one triple for each vehicle and time point. Each image in each triple is five by five pixels, where each pixel represents a cell in the vicinity of the vehicle at that particular time point. Each pixel is either black or white, where black represents a cell containing a vehicle and white represents an empty cell. Each triple consists of a center, left and a right image, where the center image represents the five-by-five cells ahead of the vehicle, the left image contains the five-by-five cells ahead of the vehicle to the left of the vehicle and the right image the five-by-five cells ahead of the vehicle to the right of the vehicle. See Figure 4 for an up-scaled example of such a triple.

Measurement	**Value**	**Comment**
Number of rows	14901	
Occurring values of lane change	$0, \pm 1$	
Number of lane changes	3	Very few lane changes occurs, making these difficult to predict.
Minimum movement	0.000 cell lengths	Occurs when traffic has come to a complete halt
Maximum movement	0.214 cell lengths	
Average movement	0.119 cell lengths	
Median movement	0.122 cell lengths	

Table 2: Measurements describing the stochastically generated data set.

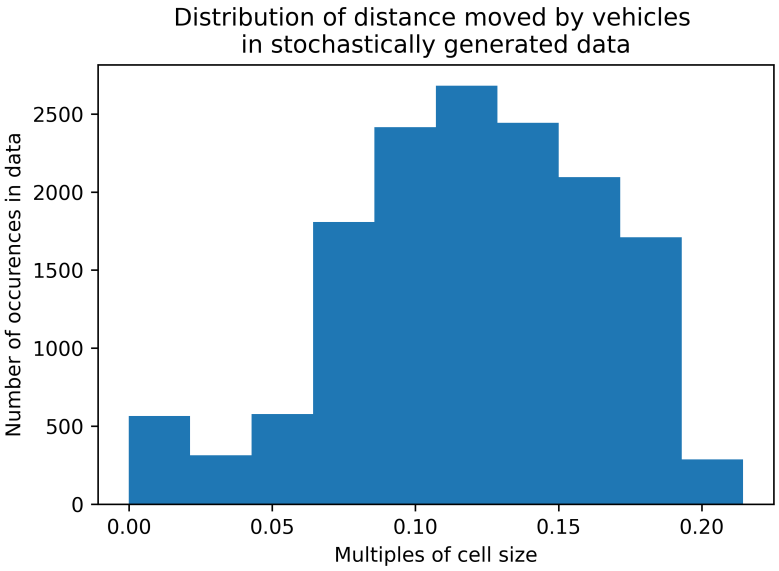

Figure 3: Histogram describing the distribution of the distances moved during each time step by vehicles in the stochastically generated data set.

Column	**Description**	**Details**
Steering	Distance traveled	Distance travelled by vehicle until next time point, measured in number of cells.
Lane	Lane change	Lane change until next time point. 0 - no change, 1 - change to the right, -1 - change to the left.

Table 3: Description of stochastically generated data set

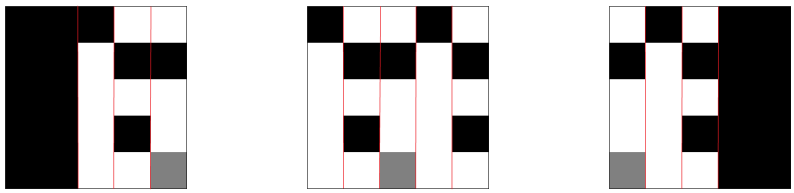

Figure 4: Enlarged examples of left, center and right images. The pixel representing the vehicle to which the images correspond is shown in grey and the borders of the lanes are shown in red. The images used as input to the CNN are black and white only, with black representing a cell containing a vehicle and white representing a cell without a vehicle. In this example triple, the vehicle is in the third lane and the left black band in the left image and the right black band in the right image represent cells outside of the road.

To make use of the spatial ordering of the input, a convolutional neural network (CNN) is used. Firstly, the hyper parameters of the CNN are determined by training on the set of stochastically generated images. After this, images corresponding to the original data set are produced, on which the CNN with the previously determined hyper parameters is trained.

3.2 Determining hyper parameters for CNN

The parameters that make up the structure of the CNN are called *hyper parameters*. These are not optimized during the training phase of the CNN, but rather decided upon prior to this phase. In this sense they are different from the weights and biases of the CNN. In order to determine the set of hyper parameters to use for the final CNN, many different sets of hyper parameters were used to create CNN:s. As this is unfeasible to achieve on a personal computer due to the large computational power needed, a computer hosted by Center for Scientific and Technical computing at Lund University was used. The computer is made up of more than 50 compute nodes paid for by Lund University research groups. Each of these nodes has two Intel Xeon E5-2650 v3 processors consisting of 20 cores each and each node has 64 GB of DDR4 RAM.

3.2.1 Using stochastically generated images

In order to determine the possibility of creating a CNN able to predict the movement of individual vehicles, a readily available synthetic data set and corresponding images derives from the original data set is used. This is done in order to not create images from the real data set unnecessarily.

3.2.2 Determining hyper parameters

In order to determine the hyper parameters, multiple sets of hyper parameters were used in training CNN:s this was achieved by using shell scripts consisting of nested loops. Each such loop would iterate over different values for a specific hyper parameter, see pseudo code in Listing 1. See the list of hyper parameters and the values considered in Table 4. Different number of hidden layers and nodes were also considered. The general layout consisted of a set of four convolutional layers, followed by a set of 3-5 dense layers, see Figure 5 for a graphical representation. Different sets of convolutional layers were considered, see Table 5 for details. Different sets of dense layers were also considered, see Table 6 for details. All of the CNN:s considered were trained for 50 epochs and the CNN from the last epoch was selected.

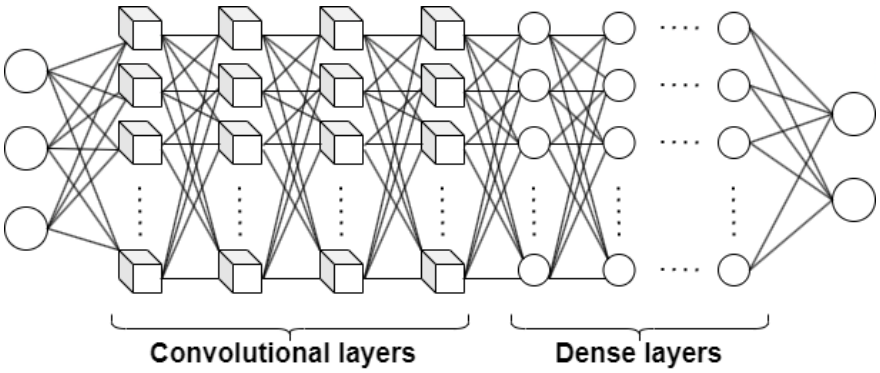

Figure 5: Graphical representation of the general layout of the CNN used.

Listing 1: Pseudo code to train multiple CNN:s

```
for batchSize in b_1, b_2, ...
    for learningRate in l_1, l_2, ...
        ...
        trainCNN with bSize = batchSize,
                      lRate = learningRate,
                      ...
    done
done
```

Hyper parameter	**Values tested**
Proportion of data used for testing	0.2, 0.5
Proportion of nodes	0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5
batch size	100, 128, 256, 64
learning rate	0.000001, 0.00001, 0.0001, 0.001, 0.01, 0.1
conv type	1, 16, 2, 4, 5, 6, 61, 62, 63, 64, 7, 8, 9
dense type	1, 2, 3, 4, 5, 6, 7, 71, 72, 73, 74, 8, 9
activation	elu, hard sigmoid, linear, relu, selu, sigmoid, softmax, softplus, softsign, tanh
optimizer	adadelta, adam, RMSprop

Table 4: Hyper parameters and the values considered when determining training a CNN based on the stochastically generated images.

Type	**Number of nodes per layer**
1	24 - 36 - 48 - 64
2	128 - 256 - 512 - 1024
3	256 - 512 - 1024 - 1600
4	49 - 72 - 96 - 128
5	30 - 42 - 54 - 70
6	40 - 52 - 64 - 80
6.1	40 - 60 - 64 - 80
6.2	40 - 52 - 70 - 80
6.3	40 - 52 - 64 - 90
6.4	40 - 52 - 64 - 85
7	64 - 76 - 88 - 104
8	80 - 94 - 110 - 130

Table 5: Layout of different sets of convolutional part of network.

Type	Number of layers	Number of nodes per layer
1	3	50 - 10 - 2
2	4	100 - 20 - 4 - 2
3	5	128 - 32 - 8 - 2
4	3	24 - 12 - 2
5	3	35 - 20 - 2
6	3	50 - 20 - 2
7	3	60 - 20 - 2
7.1	3	60 - 30 - 2
7.2	3	60 - 15 - 2
7.3	3	62 - 20 - 2
7.4	3	60 - 25 - 2
8	3	70 - 30 - 2
9	3	80 - 30 - 2

Table 6: Layout of different sets of dense part of network.

3.3 Building data set from real data

In the text file containing the real data, the parameters that are of interest of predicting, namely distance traveled until next time point and possible lane change until next time point, are not present and thus need to be calculated. Furthermore, there are no images available and so these need to be generated.

3.3.1 Creating images

Firstly, images corresponding to every vehicle at every time point are created. This is done by following the following steps for each vehicle for each time point that the vehicle is on the road:

(i) Retrieve the position of the vehicle at the next time point,

(ii) Calculate the directional vector of the vehicle,

(iii) Calculate the angle between the directional vector of the vehicle and the unit vector (0, 1),

(iv) Retrieve a list of nearby vehicles by including all vehicles within a radius of $5 \times avgVehicleLength$,

(v) Rotate the positions of the nearby vehicles, using the angle calculated in step iii, around the current vehicle's position to lay the positions of the vehicles out corresponding to the images that we want to create,

(vi) Iterate over the nearby vehicles translated positions to create a 5x5 boolean matrix, corresponding to the 5x5 cells ahead of the current vehicle, with true values corresponding to cells occupied by a vehicle,

(vii) Create images from the boolean matrix.

3.3.2 Creating database

A data set in an SQLite database is created, containing the columns described in Table 7 and measurements describing the data set in Table 8.

Column	**Description**	**Details**
Vehicle ID	Vehicle identification number	Each vehicle is identified by a unique integer.
Global time	Current time	Current time measured in milliseconds since Jan 1st, 1970.
Global X	Global lateral coordinate	Lateral coordinate according to CA State Plane III.
Global Y	Global longitudinal coordinate	Longitudinal coordinate according to CA State Plane III.
Lane change	Lane change	Lane change until next time point. 0 - no change, 1 - change to the right, -1 - change to the left.
Movement	Movement until next time point	Movement until next time point measured in average vehicle lengths.

Table 7: Description of data set generated from real data

Measurement	**Value**	**Comment**
Number of rows	1178429	
Occurring values of lane change	$0, \pm 1, \pm 2, \pm 3$	There are some errors in the original data, where vehicles are registered switching between lane 5 and auxiliary lanes 7 and 8. This explains the $\pm 2, \pm 3$.
Number of lane changes	1319	Very few lane changes occurs, making these difficult to predict.
Minimum movement	0.000 average vehicle lengths	Occurs when traffic has come to a complete halt
Maximum movement	1.456 average vehicle lengths	
Average movement	0.255 average vehicle lengths	
Median movement	0.268 average vehicle lengths	

Table 8: Measurements describing the data set generated from the original data.

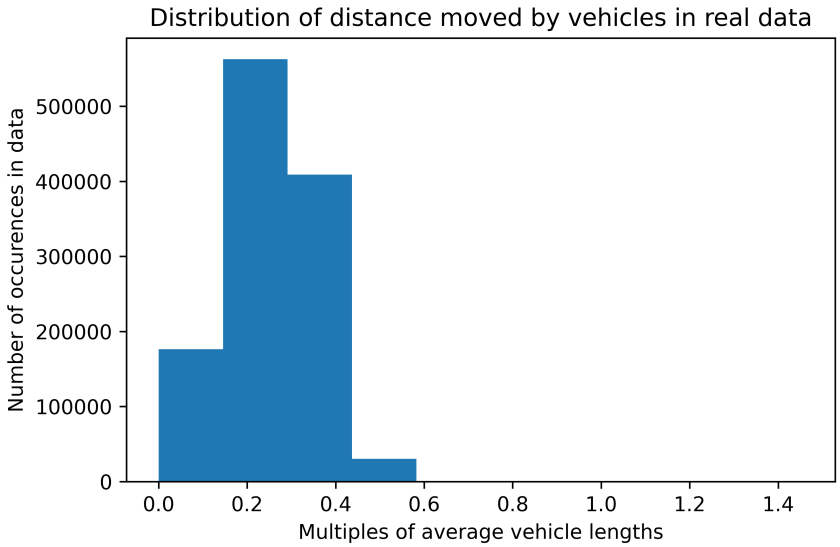

Figure 6: Histogram describing the distribution of the distances moved during each time step by vehicles in the data set generated from the original data.

3.4 Train final CNN

Using the database and images generated from the real data, the final CNN is trained with the best set of hyper parameters determined as described in section 3.2.

3.5 Extend CNN

Once a CNN has been created to predict the behaviour of individual vehicles with only images as input, this needs to be extended to simulate the behaviour of all vehicles on the road. This is done by using an initial layout of vehicles on the section of road and then move all vehicles one by one by using the CNN. The vehicles on the road section at each time point are stored in a list of lists, each list corresponding to one of the 6 non-auxiliary lanes. The processing at each time point of the simulation is described below:

(i) Add vehicles to the list corresponding to each of the lanes by retrieving the vehicles from the database that enter the road section at the current time point,

(ii) Create images for each vehicle currently on the road section,

(iii) Use the CNN to predict the movement of each vehicle,

(iv) Update the list of lists to reflect the new position of all of the vehicles,

(v) Remove vehicles if their new positions are beyond the end of the road section

4 Results

4.1 Determining hyper parameters

Out of all of the different sets of hyper parameters considered for the CNN:s trained on the stochastically generated images, the best 40 performing ones, based on the validation error, are listed in Table 9. As seen in the table, the hyper parameters which resulted in the lowest mean squared error (MSE), 0.001271, used a hold out data set consisting of 20% of the original data set, a drop out probability of 20%, four convolutional layers consisting of 40, 52, 64 and 80 nodes, three dense layers consisting of 60, 30 and 2 nodes, scaled exponential linear unit as the activation function, a batch size of 64, RMSprop as optimizer a learning rate of 0.0001.

test size	keep prob	conv type	dense type	val loss
0.2	0.2	6	7	0.001271
0.2	0.2	6	8	0.001276
0.2	0.2	7	5	0.001276
0.2	0.2	5	7	0.001281
0.2	0.2	4	9	0.001296
0.2	0.2	7	6	0.001304
0.2	0.2	7	1	0.001305
0.2	0.2	1	8	0.001307
0.2	0.2	8	1	0.00131
0.2	0.2	6.3	7.1	0.00132
0.2	0.2	6	1	0.001321
0.2	0.2	7	7	0.001322
0.2	0.2	4	6	0.001323
0.2	0.2	1	9	0.001323
0.2	0.2	1	1	0.001329
0.2	0.2	8	7	0.001338
0.2	0.3	8	9	0.00134
0.2	0.2	6	7	0.001342
0.5	0.2	1	9	0.001344
0.2	0.2	7	8	0.001348
0.2	0.2	6	4	0.001349
0.2	0.2	6.3	7.3	0.001351
0.2	0.2	4	7	0.001353
0.2	0.2	8	9	0.001354
0.2	0.2	6.3	7.4	0.001357
0.2	0.2	1	6	0.001357
0.2	0.2	6.2	7.1	0.001362
0.2	0.2	1	2	0.001364
0.2	0.2	6.4	7.1	0.001366
0.2	0.2	8	6	0.001368
0.2	0.2	7	4	0.00137
0.2	0.2	5	6	0.001374
0.2	0.2	4	4	0.001377
0.2	0.2	5	4	0.001387
0.2	0.2	6.4	7.4	0.001392
0.2	0.2	8	8	0.001393
0.2	0.2	4	5	0.001393
0.2	0.25	8	9	0.001396
0.2	0.2	6.2	7.2	0.0014
0.2	0.35	8	9	0.001408

Table 9: The 40 sets of hyper parameters for the CNN trained on the stochastically generated images resulting in the lowest validation errors. All of these sets of hyper parameters uses the scaled exponential linear unit as activation function, a batch size of 64, RMSprop as optimizer, a learning rate of 0.0001 and MSE as loss function.

4.2 Final CNN performance on individual vehicles

Using the hyper parameters set determined in the previous section, the CNN trained on the real data achieved an MSE of 0.0025897 on the hold out data set.

4.3 Model performance compared to reality

We simulate the behaviour of all vehicles on the section of road and compare their behaviour to reality, measured at the midsection for time periods $t \in \{7:50-7:55, 7:55-8:00, 8:00-8:05\}$. The measures that are compared are flow (vehicles per hour), time mean speed (TMS) and space mean speed, where TMS and SMS are defined by

$$\text{TMS}(t) = \frac{\sum_i v(t)_i}{n(t)} \tag{26}$$

and

$$\text{SMS}(t) = \frac{\sum_i d(t)_i}{\sum_i tt(t)_i} \tag{27}$$

where $v(t)_i$ is the velocity of vehicle i during time period t measured at midsection, $n(t)$ is the number of vehicles passing the section of road during time period t, $d(t)_i$ is the distance traveled by vehicle i on the section of road during time period t and $tt(t)_i$ is the time traveled by vehicle i during time period t. The result can be seen in figures 7, 8 and 9.

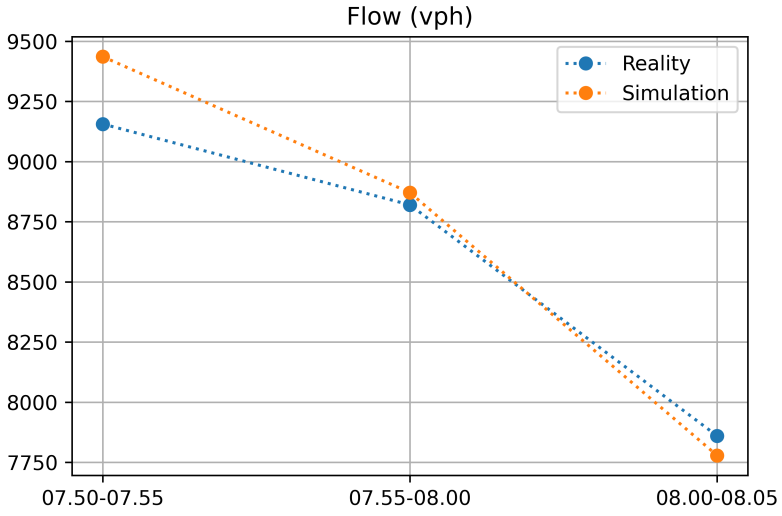

Figure 7: Comparison of flow between simulation and reality.

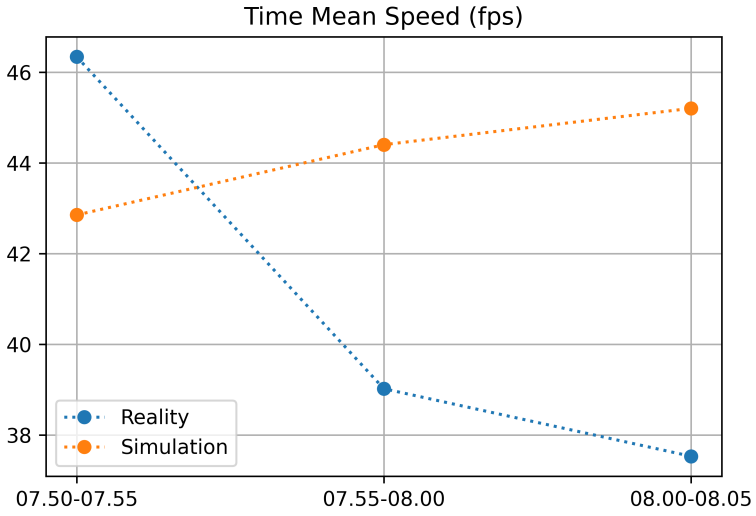

Figure 8: Comparison of TMS between simulation and reality.

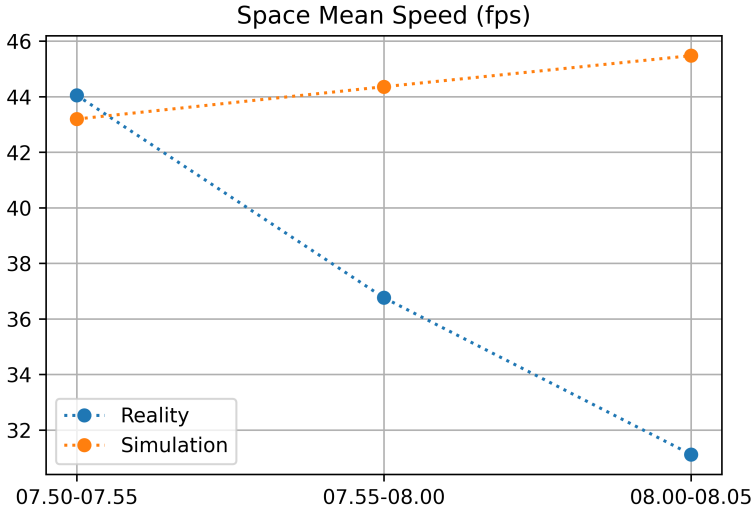

Figure 9: Comparison of SMS between simulation and reality.

5 Discussion and conclusion

5.1 Discussion of results

As described in Section 4.1, it was possible to determine a set of hyper parameters for the CNN trained on the stochastically generated data that resulted in an MSE of 0.001271 on the hold out data set (Table 9). While the MSE depends on both movement and lane change, we focus on how this relates to the movement as the lane changes are so unusual. When compared to the median movement of 0.122 cell lengths (Table 2), we realise that the performance is quite poor as the predicted value on average differs 0.035651 from the real value. If we assume that we predict the lane change perfectly and that the real movement of a vehicle is the median movement, this corresponds to a 29% error. Regardless of this, a CNN was trained using the same hyper parameters but with images and data generated from the real data set.

This achieved an MSE of 0.0025897 on the hold out data set and when compared to the median movement 0.268 average vehicle lengths in the real data (Table 8) we see that this performs relatively better using the same reasoning as in the previous paragraph, but still results in an average error of 19%.

It is likely that it would be possible to determine hyper parameters which would result in a better performing CNN. However, it is even more likely that a CNN could predict the movement of vehicles even better if time sensitivity would be included in the model in some way. This could be done in different ways, possibly by using a different type of CNN, such as a convolutional LSTM or by changing the colors of the cells in the images based on the velocity of the vehicles represented by each such cell.

It can be seen that the flow predicted by the model is similar to the real flow (Figure 7). This is to be expected, as the input to the model apart from the initial layout of vehicles on the stretch of road is time of vehicles entering the stretch of road.

For the first five minutes of the simulation the TMS (Figure 8) and SMS (Figure 9) are somewhat acceptable. However, there are big discrepancies between the simulated and actual values for both these measurements for the following two time periods. While the actual values of both decreases as the traffic flow reduces, the simulated values increases. This is likely the result of errors propagating. If the movement of the vehicles are predicted to be larger in the simulation than in reality, this would result in a less dense layout of vehicles on the stretch of road, in turn resulting in even larger predicted movements of the vehicles.

5.2 Possible improvements to method

Utilize an existing framework to determine hyper parameters for CNN. There are different methods available to automatically determine hyper parameters, such as Bayesian model selection [RW05]. This approach to finding suitable hyper parameters is likely to be more successful than the somewhat random approach taken in this project.

Include the time sensitivity of the data as input to the CNN. The method used does not take the temporal aspect of the data into consideration, as the objective was to only use images as input to ANN. However, it is likely that a model which use this information as input would be more accurate. It would have been possible to use the information in the original data set to color the pixels in the images differently, depending on the velocity of the corresponding cars and still use a traditional CNN, or a convolutional LSTM network could be utilized.

Use separate CNN:s to predict the movement of a vehicle and the probability of a lane change, respectively. The CNN:s used in the project were trained to predict both the movement as well as lane change of a vehicle until next time point. This has two drawbacks.

(i) If the movement of a vehicle be denoted by x and the lane change of a vehicle be denoted by i. Then $x \in \mathbb{R}^+$ and $i \in \{-1, 0, 1\}$. However, as the scaled exponential linear unit activation function used in the output layer, the output vector $v \in \{(u_1, u_2) \in \mathbb{R}^2 | u_1, u_2 \geq -1\}$. It would be reasonable to use an activation function designed for classifications problems for the lane change output, which could be achieved by the use of two separate ANN:s.

(ii) In approximately 99.9% of the cases, no lane change occurs. Thus, always predicting no lane change will have practically no negative impact on the cost function. This suggest that some under-sampling technique should be used when training this separate network in order to more accurately predict lane changes.

Dynamically calculate constants. There are a number of constants which occur in multiple programs. These were originally calculated from the database created from the original data, and then copied to the programs in which they were used. This did not cause any errors, but this type of copying and pasting of information is of course error prone.

Use a remote GIT repository for version control. As the project included many iterations of many programs it would have been beneficial to use GIT for version control. Since the project spanned over some time, the files used were moved about to different folders on different computers and different

cloud hosting services. Less time would have been spent on gathering all these files if a remote GIT repository would have been used.

References

[Ben91] Yoshua Bengio. "Artificial Neural Networks and Their Application to Sequence Recognition". UMI Order No. GAXNN-72116 (Canadian dissertation). PhD thesis. CAN, 1991.

[RW05] Carl Edward Rasmussen and Christopher K. I. Williams. *Gaussian Processes for Machine Learning*. The MIT Press, 2005. DOI: 10.7551/mitpress/3206.001.0001. URL: https://doi.org/10.7551/mitpress/3206.001.0001.

[GBB11] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. "Deep Sparse Rectifier Neural Networks". In: *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*. Ed. by Geoffrey Gordon, David Dunson, and Miroslav Dudík. Vol. 15. Proceedings of Machine Learning Research. Fort Lauderdale, FL, USA: PMLR, Nov. 2011, pp. 315–323. URL: https://proceedings.mlr.press/v15/glorot11a.html.

[DS14] Ke-Lin Du and M. N. S. Swamy. *Neural Networks and Statistical Learning*. Springer London, 2014. DOI: 10.1007/978-1-4471-5571-3. URL: https://doi.org/10.1007/978-1-4471-5571-3.

[CUH15] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2015. DOI: 10.48550/ARXIV.1511.07289. URL: https://arxiv.org/abs/1511.07289.

[ON15] Keiron O'Shea and Ryan Nash. *An Introduction to Convolutional Neural Networks*. 2015. DOI: 10.48550/ARXIV.1511.08458. URL: https://arxiv.org/abs/1511.08458.

[Rud16] Sebastian Ruder. *An overview of gradient descent optimization algorithms*. 2016. DOI: 10.48550/ARXIV.1609.04747. URL: https://arxiv.org/abs/1609.04747.

[Sch16] Benjamin J. E. Schroeter. "Artificial Neural Networks in Precipitation Nowcasting: An Australian Case Study". In: *Artificial Neural Network Modelling*. Springer International Publishing, 2016, pp. 325–339. DOI: 10.1007/978-3-319-28495-8_14. URL: https://doi.org/10.1007/978-3-319-28495-8_14.

[Sha16] Subana Shanmuganathan. "Artificial Neural Network Modelling: An Introduction". In: *Artificial Neural Network Modelling*. Springer International Publishing, 2016, pp. 1–14. DOI: 10.1007/978-3-319-28495-8_1. URL: https://doi.org/10.1007/978-3-319-28495-8_1.

[Kla+17] Günter Klambauer et al. "Self-Normalizing Neural Networks". In: (2017). DOI: 10.48550/ARXIV.1706.02515. URL: https://arxiv.org/abs/1706.02515.

[Sil+17] Ivan Nunes da Silva et al. *Artificial Neural Networks*. Springer International Publishing, 2017. DOI: 10.1007/978-3-319-43162-8. URL: https://doi.org/10.1007/978-3-319-43162-8.

[DSC21] Shiv Ram Dubey, Satish Kumar Singh, and Bidyut Baran Chaudhuri. *Activation Functions in Deep Learning: A Comprehensive Survey and Benchmark*. 2021. DOI: 10.48550/ARXIV.2109.14545. URL: https://arxiv.org/abs/2109.14545.

[22] *Keras Python library*. https://keras.io/. Accessed: 2022-11-27. 2022. URL: https://keras.io/.

Master's Theses in Mathematical Sciences 2022:E76
ISSN 1404-6342

LUNFNA-3037-2022

Numerical Analysis
Centre for Mathematical Sciences
Lund University
Box 118, SE-221 00 Lund, Sweden

http://www.maths.lu.se/