Vehicle Motion Prediction in Autonomous Vehicles using an FCN-based Deep-Learning Model

Sarvesh Bhalchandra Telang
Commercial Vehicle Technology
Technical University Kaiserslautern
Kaiserslautern, Germany
telang@rhrk.uni-kl.de

Nilay Gawde
Commercial Vehicle Technology
Technical University Kaiserslautern
Kaiserslautern, Germany
gawde@rhrk.uni-kl.de

Kaivalya Patkar
Commercial Vehicle Technology
Technical University Kaiserslautern
Kaiserslautern, Germany
patkar@rhrk.uni-kl.de

Abstract— **Vehicle motion prediction is an important part of ADAS (Advanced Driver Assistance Systems), used to predict the future trajectory of the vehicle. The use of deep-learning models is becoming increasingly relevant to predict trajectories in complicated scenarios. These models need to be trained on a large and scenario appropriate dataset to attain accurate results that are close to the ground truth. Risk of accidents increases at higher vehicle speeds and more complex road geometries. Dependence only on vehicular sensors in such scenarios is dangerous due to sensor occlusions. In our work, we consider inD (Intersection Drone Dataset) which provides naturalistic vehicle trajectories. A birds-eye-view prevents any occlusion within the intersection. Through the study of various motion prediction models, a problem definition for our model was set. The use of FCN keras model is beneficial for computer vision tasks and thus it was selected for our approach.**

Keywords— ADAS, Deep Learning, FCN Keras

I. Introduction

Improving the road safety is an important field of study which is being supported by a lot of key research. In order to avoid such accidents, stricter traffic rules and more stringent driver's licence tests are being implemented. However, in certain situations, weak reaction time from the driver result into an accident. Such human errors can be avoided by Advanced Driver Assistance systems and Autonomous Vehicles [1]. For these systems to work safely on the road, they need to be actively aware of the surroundings (other vehicles, pedestrians, bicycles) and also be able to predict their future manoeuvres [2]. Each future prediction in any situation comes with a certain amount of risk [1]. The measure of this risk can be given through various mathematical models which are discussed by [1]. The author [1] categorises these model into:

1) *Physics-based motion models*
2) *Manoeuvre-based motion models*
3) *Interaction-aware motion models*

Each model is more complex than the former, thus increasing the prediction results. Other method to predict the trajectories of these vehicles are by using Neural Networks (machine learning techniques). These techniques can be used to train a large dataset on a specific neural network model. This trained model is then used to test on similar road conditions to predict the trajectory of the vehicle. Neural Network approaches are gathering more attention due to its capability of producing more accurate results than mathematical models in more realistic and complex scenarios [2]. The authors of [2] divides the entire model into three categories.

1) *Input Representation:* The type of input data and its representation for the model differs [2]. The data can be raw sensor data of the target vehicle or from a bird's eye view [2].

2) *Output type:* It signifies the form of vehicle behaviour that the prediction model gives [2].

3) *Prediction Method:* This provides the type of Prediction model and the Neural Network being used. These can be among recurrent neural networks, convolution neural networks, graph neural networks.

While predicting the path of vehicle is critical, being aware of other road users and predicting their future trajectory is also important to avoid any accidents. The works [3–5] demonstrate various methods to detect the human motion and predict their trajectories. This can be clubbed with the vehicle motion prediction models to predict the collisions between vehicles and pedestrians, especially at intersections. However, in this paper we will be only focusing on the vehicle motion prediction models.

The works of [6], [7] study the interactions of the road users at intersections. Most of the accidents occur at the intersections, hence, there is a need to develop vehicle prediction models that are accurate and take less computational time to produce the prediction results. The paper [7] highlights the importance of short computational time to evaluate the prediction results. This is very important as a highly accurate result which high computational time isn't beneficial to take precautionary measures before an impending collision.

In this paper we use Neural Networks to develop an FCN keras-model. The input data is the trajectory dataset of German intersections called inD (intersection Drone dataset) [8].

Firstly, understanding various mathematical models and Neural Network models is important to understand the advantage of each technique. We will be using certain common vehicle terminologies while studying these techniques.

B. Terminologies used for vehicle behaviour [2, p. 34]

1) *Target Vehicles (TVs):* These are vehicles whose behaviour we are interested in predicting.

2) *Ego Vehicle (EV)*: It is the autonomous vehicle which observes the surrounding environment to predict the behaviour of the TVs.

3) *Surrounding Vehicles (SVs):* These are the vehicles whose behaviour is explored by the prediction model as it can potentially impact TV's future behaviour. Different studies may adopt different criteria for selecting SVs based on their Modelling assumptions.

4) *Non-Effective Vehicles (NVs):* These are the remaining vehicles in driving environment that are

assumed to have no impact on the TV’s behaviour. [2, p. 34]

C. *Vehicle motion models*

The mathematical models-based prediction methods are less complex than methods based on neural networks. These are methods that require short computational time. However, they produce less accurate results.

1) Physics-based motion models: Physics-based motion models are most studies and thus are the most implemented prediction models [1]. These models are only concerned with the laws of physics and the motion prediction depends on input parameters like acceleration, steering angle, speed of vehicle, and weight [1]. This model relies on the dynamic and kinematics model; hence, the proper implementation of road geometries improves the trajectory prediction accuracy [1]. The kinematic model considers the mathematical relationship between velocity, displacement, and acceleration of the vehicle [1]. Examples of physics-based motion models using the kinematic model are Constant Velocity Model, Constant Acceleration Model, Constant Turn-rate and velocity model, and Constant Turn-rate and Acceleration model [1]. However, physics-based motion models are limited by a short prediction span of less than a second in the future [1].

2) Manoeuvre-based motion models: Manoeuvre-based motion models consider the deliberate manoeuvre motion from the driver into its algorithm to the trajectory of the EV [1]. This is done independently from the motion of SVs [1]. The model uses training data which provides us with clustered trajectories of the on-road vehicles [1]. These trajectories are more reliable than the predictions by physics-based models [1]. The drive’s manoeuvre intentions can be estimated by factors such as drivers motion, the physical state of the vehicle, and the data of the

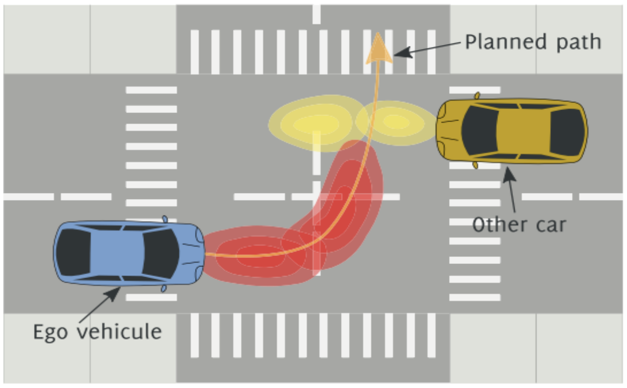
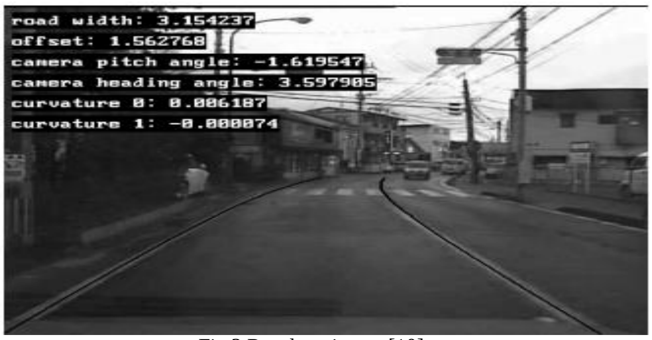


Fig.1 Manoeuvre-based motion model [1].

road network [1]. However, introduction of large variations requires several prototypes which increases the time required to evaluate the trajectories of the vehicles [1]. The Fig.1 depicts the basic idea of this motion model. Impractical trajectories generated by the model are handled by providing the trajectories and the current state of the vehicle, generated from the Gaussian Processes, as inputs for the Rapidly-exploring Random Tree (RRT) [1], [9].

3) Interaction-aware motion models: This method considers the presence of other vehicles on the road and their trajectories, to predict the behaviour of the EV [1]. Trajectory prototypes of these SVs are key for Interaction-aware model [1]. However, it is impractical to track all the interdependencies of the vehicles on road [1]. Interaction-aware motion models generally use Dynamic Bayesian Networks (DBN) [1]. The model works on the assumption that the SVs are an hindrance for the EV’s trajectory [1]. Interaction-aware motion models are benefited with higher reliability rates and are able to predict farther in future [1]. They are limited by the computational effort and time [1].

Fig.2 Location on the map [10].


```
road width: 3.154237
offset: 1.562768
camera pitch angle: -1.619547
camera heading angle: 3.597905
curvature 0: 0.006187
curvature 1: -0.000074
```
Fig.3 Resultant image [10].

D. *Prediction techniques based on Neural Networks*

1) Recurrent Neural Networks(RNN): “The simplest RNN can be considered as an extension to two-layer fully-connected neural network where the hidden layer has a feedback” [2, pp. 39-40]. Gated RNNs are used to train the network to learn long sequences in order to approximate them to sequence mapping [2]. Example of gated RNN are Long-short term memory (LSTM) and Gated recurrent unit

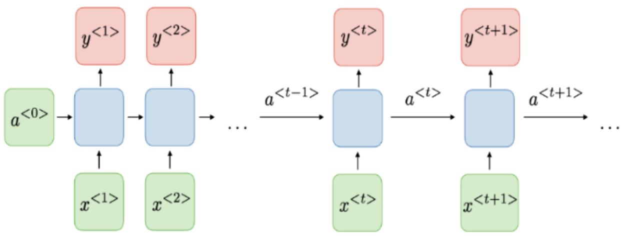


Fig.4 Standard RNN structure [11].

2) (GRU) [2]. They contain gated architecture inside each cell rather than a simple fully connected hidden layer

[2]. However, RNN struggle in modelling vehicles spatial interaction and image data [2].

3) Convolutional Neural Netwoks (CNN): CNN are designed to reduce the spatial size of the input by sub-sampling [2]. This is done by placing learnable weights on the inputs within the convolution layers [2]. RNN's struggle to deal with image data can be solved by using CNNs as they can extract data from images [2]. When dealing with a Birds-eye-view image, a convolution-deconvolution architecture could be used for image segmentation [2]. The convolutional network in the model generates a feature network and the deconvolutional network can upscale it to the output image [2]. In our work we will be using FCN-Keras model, which is a fully convolutional Neural Network. While predicting vehicle behaviour, the use of CNN is beneficial as the accepting input and generating output of image-like data is possible [2]. It also maintains the spatial relationship of the input data while processing it [2]. These benefits allows us to model vehicle interactions, driving scene context, and to produce occupancy map output [2].

4) Combination of RNNs and CNNs: This method is useful as it combines the temporal feature extracting power of RNNs and the spatial feature extracting power of CNNs [2].

5) Graph Neural Networks: Vehicles and interaction among them are considered in this method, where the vehicles are assumed as the nodes of a graph and the interactions as its edges [2]. Graph-based interaction-aware trajectory prediction model (GRIP) is one such model that is discussed by [2].

6) Encoder-Decoder CNN: The Semantic Segmentation technique uses a simple method of encoder-decoder CNN for effective outputs. The network is built after the complete segmentation processing of the input image, a crossing in our case. A decoder can be used as a binary output channel for detection. The other branch can carry out the embedding functions. Curve-fitting and clustering is an important step for the output obtained which is usually a feature map. If there is a lack of substantial amount of marked data for training, then real-time segmentation technique can be a prudent option. [12]

7) LSTM Network: Modelling multiple sequential frames as a time-series, RNN can accept the feature map whereas CNN provides the input. A special type of RNN i.e., Long Short-Term Memory and Gated Recurrent Unit

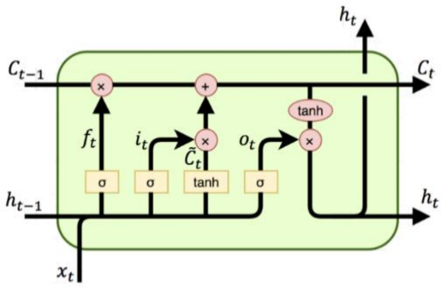


Fig.4 Standard LSTM Network [13]

is proposed. LSTM has a speciality of forgetting irrelevant information over mainstream RNN. A double layered LSTM or Convolutional LSTM can also be employed for error reduction.

8) Convolutional LSTM: Analysing time-based information is the speciality of Long Short Tem Memory (LSTM) model of the deep learning. It is also considered as the basic building block in the domain of real-time video computation. It has a useful structure of feedback loop for powerful extraction of image features. Convolutional LSTM or ConvLSTM has an etraordinary input to output ration in the domain of semantic video segmentation. In this particular field, the ConvLSTM work efficiently where the input is dependent on time and confined to one plane only. [12]

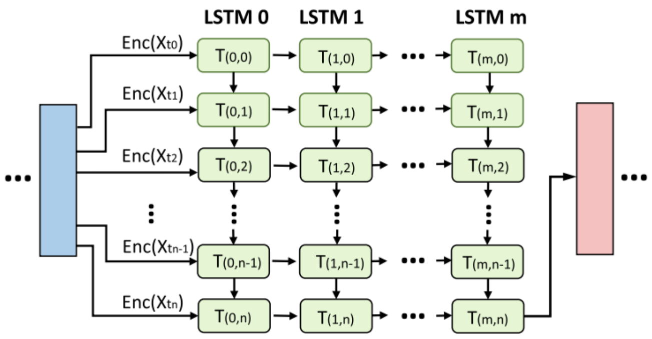


Fig.5 Convolutional LSTM Network Image as input and feature map as output [12]

II. Implementation

A. Selection of training data

In this paper, we have considered an inD dataset (Intersection Drone Dataset) from [8]. This dataset contains naturalistic vehicle trajectories recorded at intersections in Germany [8]. Occlusions, which are common limitations of traditional traffic data collection methods, are overcome by using a drone [8]. The dataset is primarily divided into four distinct locations that contain traffic data. For our implementation, we chose one of the location sets, which included a total of 12 recordings. Five of the twelve recordings are utilized for the training purpose based on the characteristics of the data. Each recording data set contains parameters such as the number of traffic objects observed, the number of frames and frame rate, the x and y positions of traffic objects, the velocity and acceleration in the x and y directions, the width and length of the traffic objects.

B. Data Preprocessing and normalization

Data pre-processing is a method for converting raw data into a comprehensible and meaningful format [14]. When raw data is gathered from certain sources, pre-processing is necessary to make accurate predictions using neural networks [14]. First, noisy, unnecessary, and large unstructured data is removed using pre-processing. Then, down-sampling is performed to shorten the sample time and save RAM usage. The next stage is applying min-max scaling to normalize the data. The technique of scaling attribute values to fit inside a certain range is known as normalization [15]. All values are converted into decimal numbers between 0 and 1, with the minimum and maximum values of each attribute becoming 0 and 1, respectively [15]. As mentioned in [15], the formula for this is given below:

$$normalized\ (x) = \frac{minRange+(xmin\ value)(maxRange - minRange)}{maxValue - minValue} \quad (1)$$

C. Data Preparation

Now we need to prepare the data for training and testing purpose. A sliding window technique is used to approximate the time series data's actual value [14]. The window and segment sizes grow until we find the closest estimate with the least error [14]. The subsequent segment is chosen after the end of the first segment. The procedure is carried out repeatedly until all-time series data have been segmented [14]. Figure 6 depicts the sliding window method with window size of 5 [14].

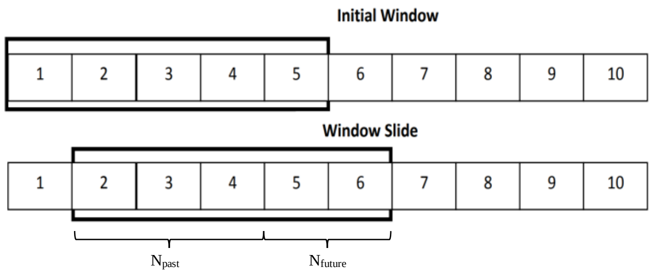


Fig.6 Sliding window on a time series [14]

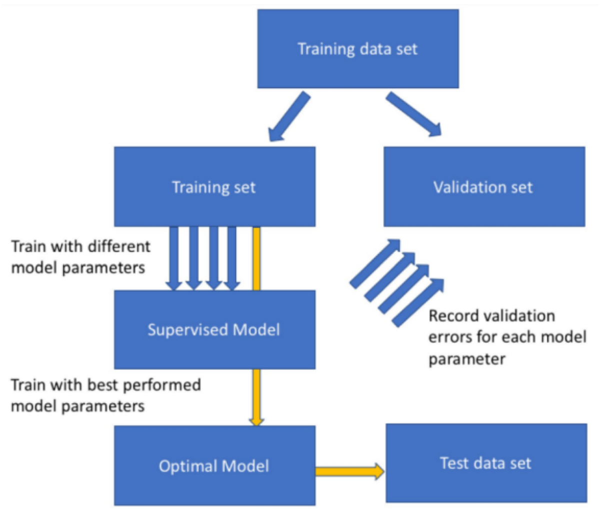
The output y should be predicted by the prediction function using some of the data from the past values x. The dataset can be used to determine which y to predict for each value of x. Therefore, at time step t, we need to know our current data point x_i (which could be velocity or position or anything else) as well as a history of data points $x_{i-1}\ to\ x_{i-N_{past}}$ from previous time steps (N_{past} = 3). When we train a neural network, we aim to approximate the function that converts our inputs x to our outputs y. The function can be described as follows:

$$y = f(\,[\,x_1^T\ x_2^T \ldots x_n^T\,]\,) \quad (2)$$

Therefore, in order for our training procedure to be able to determine how well the target function is approximated, we must first know how our predictions y should appear. In this instance, we also need to grab the future values $x_{i+1} \ldots x_{i+N_{future}}$ and declare them as output values $y_{i+1} \ldots y_{i+N_{future}}$ (N_{future} = 2). As a result, we will predict $\hat{y}_{i+1} \ldots \hat{y}_{i+N_{future}}$ and compare it to the actual $y_{i+1} \ldots y_{i+N_{future}}$, which is something we must do in order to assess how well we did with our prediction.

D. Train test split

Model complexity increases the model's discriminating power, but it also increases the likelihood of over-fitting [16]. Over-fitting is a condition that occurs frequently when a trained model performs incredibly well on the training examples but performs poorly on new, untrained samples, or when the model does not generalize well [16]. Splitting the data into a training set and a validation set is important to determine the model parameter(s) that best balance these two factors [16]. Each trained model is tested against the validation set after being built with a variety of model parameter settings using the training set [16]. The validation set includes samples with known origin, but the model is unaware of these classifications; as a result, predictions on the validation set enable the operator to evaluate model performance [16]. This is known as the model selection method, and it is illustrated in the Fig. 7 [16] below. Based on the errors on the validation set, the optimal model parameter(s) set is determined using the one with the lowest validation error [16].



Training data set
Training set
Validation set
Train with different model parameters
Supervised Model
Record validation errors for each model parameter
Train with best performed model parameters
Optimal Model
Test data set

Fig.7 Typical flowchart for model selection method. The validation process is represented by blue arrows, while the final training and test on blind test set process is represented by yellow arrows [16].

Consider the Fig. 3 [17] below to gain a better understanding of the overfitting phenomenon. Overfitting is a result of the model being too complex and having too many parameters [17]. Hence, the model remembers the training set in an excessively complicated manner rather than learning it [17]. As a result, while having decent training data fit, the prediction effect and generalization ability for the new data in the prediction set are poor.

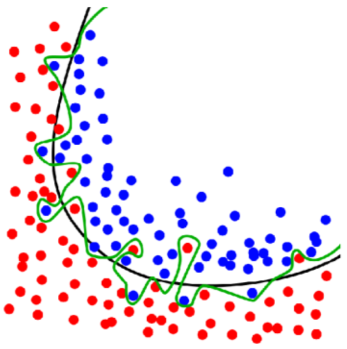
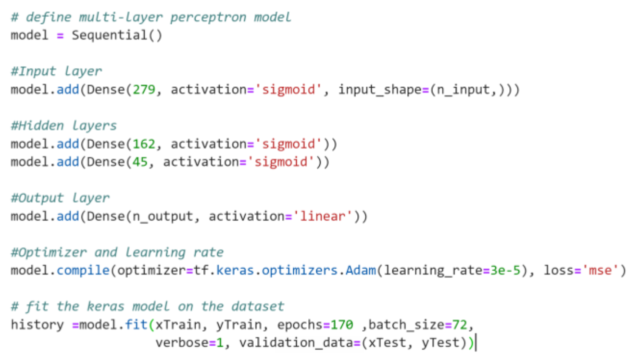


Fig.8 The overfitting model is shown by the green line and the normal fit model by the black line [17].

In our case, the_data_prepare_obj collects the vectors from the time series in the sliding window form using the function data stacking(). Additionally, it creates a single vector by stacking every sequence that was obtained underneath one another. Additionally, it splits the final, single large vector into a set of train data and test data.

E. FCN Keras Prediction Model

The FCN (Fully Convolutional Network) model is implemented with Keras on a single NVIDIA GeForce GTX1650 GPU, an AMD Ryzen 5 Mobile 5500U Processor with 8 GB memory. The Adam optimizer is used to train it for about 170 epochs with a batch size of 72. The learning rate is set at $3 * 10^{-5}$. Two hidden layers, one input layer, and one output layer were employed in this deep learning model. The final multi-layer perceptron model is shown in the Fig. 9 below.


```
# define multi-layer perceptron model
model = Sequential()

#Input layer
model.add(Dense(279, activation='sigmoid', input_shape=(n_input,)))

#Hidden layers
model.add(Dense(162, activation='sigmoid'))
model.add(Dense(45, activation='sigmoid'))

#Output layer
model.add(Dense(n_output, activation='linear'))

#Optimizer and learning rate
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=3e-5), loss='mse')

# fit the keras model on the dataset
history =model.fit(xTrain, yTrain, epochs=170 ,batch_size=72,
                   verbose=1, validation_data=(xTest, yTest))
```

Fig. 9 Final perceptron model

The most effective neural network is identified using tuning of hyper parameters and multiple iterations of testing. The loss function decreases as the number of epochs increases. The lower learning rate slightly improved the overfitting phenomenon, but the effect was not significant. Finally, we made changes in the activation function for each layer.

1. *Activation functions:* We used a non-linear sigmoid function for the top three layers while choosing linear activation function for the output layer. Below is an explanation of how the activation function works. As shown in Fig. 10 [18], the binary classification process can be carried out with ease for a single layer perceptron [18].

However, regardless of the combination, a classifier cannot solve a non-linear system's classification problem because it is fundamentally a linear equation [18]. Hence, the activation function is included in the perceptron model, as shown in Fig.11 [18]. According to [18], from the Fig. 10 [18] y_1 can be given by the equation (3) below,

$$y_1 = w_1x_1 + w_2x_2 + b \tag{3}$$

and y from the Fig.11 [18] can be given by the equation (4) and (5) below,

$$y_2 = w_1x_1 + w_2x_2 + b \tag{4}$$

$$y = \sigma(y_2) \tag{5}$$

The classification issue of the non-linear system can be solved by the model with an activation function using Equations (4) and (5) [18].

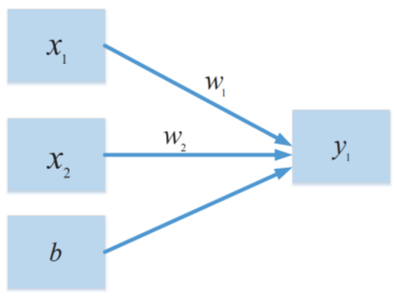


Fig. 10 an individual layer perceptron without activation function [18].

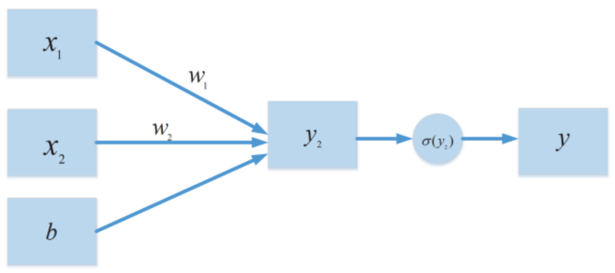
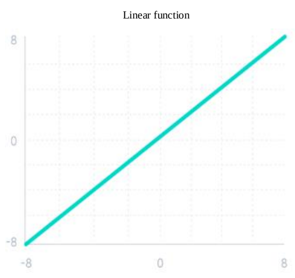


Fig. 11 an individual layer perceptron with an activation function [18].

As previously stated, we used two activation functions in our perceptron model: Sigmoid for the input and hidden layers while linear activation function for the output layer. The linear activation function (as seen in the Fig.12 [19]) can also be known as identity (multiplying by 1) or simply 'no activation function', since the linear activation function simply returns the value without altering the weighted sum of the input [19]. This function, however, has two significant drawbacks: Backpropagation cannot be used since the function's derivative is a constant and has no connection to the input x [19]. The last layer of the neural network will always be a linear function of the first layer, regardless of the number of levels [19]. Thus, a linear activation function effectively reduces the neural network to a single layer [19]. As a result, it is the most commonly used activation function for the output layers of a regression problem. It has the following mathematical representation:

$$f(x) = x \tag{6}$$

Fig.13 [18] depicts the sigmoid function curve (S-shaped), a typical non-linear activation function.



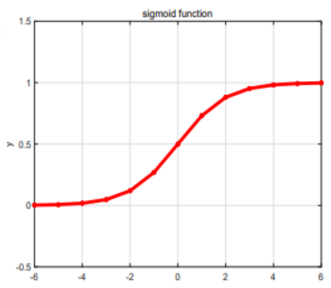


Fig.12 linear function curve [19] Fig.13 sigmoid curve [18].

Earlier, this function, which has a bounded output, was frequently utilized as the activation function [18]. Although its characteristics are consistent with the interconnections of neurons and its derivative is easy to get, the function is no longer frequently utilized due to its limitations. The gradient that is passed to the network becomes quite small when the slope of the curve is near to zero, making it difficult to successfully train the network parameters [18]. As stated in [18], the formula for the sigmoid function can be given by,

$$f(x) = \frac{1}{1+e^{-x}} \tag{7}$$

We experimented with various non-linear activation functions for the hidden layers such as Tanh, Softmax, Swish, and even the most popular ReLU variants such as ReLU, Leaky ReLU, and Parameterised ReLU. However, the sigmoid function has performed admirably for our model.

2. Optimizer and Learning rate: The optimizer can be described as an algorithm that causes the objective function to steadily decrease during the backpropagation training phase [17]. There are different optimizers for various issues [17]. The key hyperparameter for training the model is the learning rate [17]. The model requires multiple iterations to converge to the best solution when the learning rate is too low [17]. However, the gradient will repeatedly bounce around the ideal solution and may even fail to converge when the learning rate is excessively high [17]. Hence, the ideal solution should be reached with a suitable learning rate that converges steadily and quickly [17]. After experimenting with a number of well-known optimizers, Adam was discovered to have reasonably positive effects on the model. Adam combines adaptive functionality with the traits of Adagrad and RMSProp [17]. The primary advantage of employing Adam is that it uses less memory [17]. Adam has its own hyper parameters, such as learning rate, beta 1, beta 2, and epsilon. Adam has its own set of hyperparameters, including learning rate, beta 1, beta 2, and epsilon. While working on the Random search hyperparameter tuner, we discovered the optimum range for the learning rate while keeping the other parameters at their default values. Finally, the learning rate of 3e-05 produced the best results for us.

F. Prediction Testing

It is crucial to accurately estimate how well the trained and optimized model performs in general on unknown samples or blind data set and how well it generalizes [16]. Earlier, it was widely believed that the model's measured performance using the validation set was an accurate estimate of the performance of similar models [16]. However, a single split of the training and test set might yield inaccurate assessment of model performance [16]. Therefore, in this implementation, we used one of the recordings as an additional blind test set that was not used during the training or validation process to get a better estimate of the model's generalization performance. This test data set is first pre-processed and down sampled to match with the training dataset. Then, similar to the previous steps, data normalization and data stacking are performed.

G. Collecting Ground Truth and performing final evaluation

Finally, the model is validated against ground truth to determine actual performance and deviation from ideal values. It is accomplished by utilizing the three-error metrics mentioned below:

1. Average Displacement Error (ADE): It refers to the mean square error (MSE) over all estimated points of every trajectory and the true points.

$$\text{ADE} = \frac{\sum_{i=1}^{n}\sum_{t=T_{frame}}^{T_{pred}}\left[\left(\hat{x}_i^t - x_i^t\right)^2 + \left(\hat{y}_i^t - y_i^t\right)^2\right]}{n\left(T_{pred} - \left(T_{frame}+1\right)\right)} \qquad (8)$$

2. Final displacement error (FDE): FDE means the distance between the predicted final destination and the true final destination at the T_{pred} time.

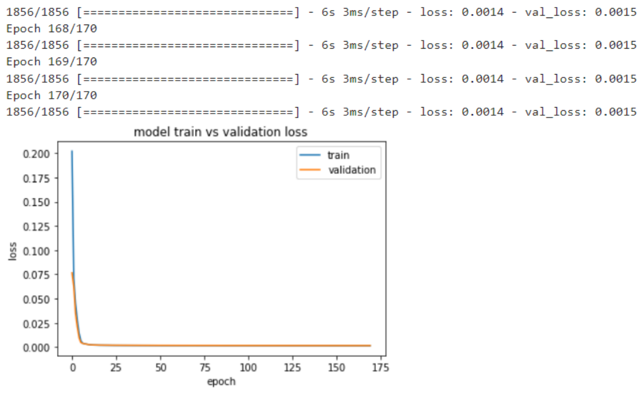
$$\text{FDE} = \frac{\sqrt{\left(\hat{x}_i^{T_{pred}} - x_i^{T_{pred}}\right)^2 + \left(\hat{y}_i^{T_{pred}} - y_i^{T_{pred}}\right)^2}}{n} \qquad (9)$$

3. Average Absolute Heading Error (AHE): This is a bit like ADE but we take the 1-norm of the error and we only consider the heading prediction here.

$$\text{AHE} = \frac{\sum_{i=1}^{n}\sum_{t=T_{frame}}^{T_{pred}}\left|\hat{y}_i^t - y_i^t\right|}{n\left(T_{pred} - \left(T_{frame}+1\right)\right)} \qquad (10)$$

III. Results and Discussion

As a result of the aforementioned implementation, our trained model successfully eliminated the overfitting and underfitting phenomena, yielding the final ADE, FDE, and AHE values of 1.004 m, 0.888 m, and 5.5 degrees, respectively. The figure 14 below shows the graph of model train against validation loss. The validation loss shows how well the model fits new data, while the training loss shows how well it matches training data. Overfitting occurs when the validation loss exceeds the training loss, whereas underfitting occurs when the validation loss is less than the training loss. According to the results, the validation loss is approaching zero and the curves are nearly matching.



```
1856/1856 [==============================] - 6s 3ms/step - loss: 0.0014 - val_loss: 0.0015
Epoch 168/170
1856/1856 [==============================] - 6s 3ms/step - loss: 0.0014 - val_loss: 0.0015
Epoch 169/170
1856/1856 [==============================] - 6s 3ms/step - loss: 0.0014 - val_loss: 0.0015
Epoch 170/170
1856/1856 [==============================] - 6s 3ms/step - loss: 0.0014 - val_loss: 0.0015
```

Fig. 14 Graph of Model Train vs Validation loss

IV. Conclusion

In this work of deep learning model for vehicle motion prediction for autonomous vehicles, a fully convolutional network (FCN) play a vital role in predicting the output for the proposed model. By studying various neural network models such as, CNN, RNN, FCN, and LSTM, the final inference led to a substantial high efficiency in FCN with respect to real time image analysis. Activation functions such as sigmoid and linear for hidden layers and output layers respectively helped in optimisation of average displacement error, final displacement error and average absolute heading error. Use of Adam optimizer and learning rate of 3e-5 were finalised after a few iterations based on observable reduction in error.

V. References

[1] S. Lefèvre, D. Vasquez, and C. Laugier, “A survey on motion prediction and risk assessment for intelligent vehicles,” *Robomech J*, vol. 1, no. 1, pp. 1–14, 2014. doi: 10.1186/s40648-014-0001-z. [Online]. Available: https://robomechjournal.springeropen.com/articles/10.1186/s40648-014-0001-z?ref=https://githubhelp.com

[2] S. Mozaffari, O. Y. Al-Jarrah, M. Dianati, P. Jennings, and A. Mouzakitis, “Deep Learning-Based Vehicle Behavior Prediction for Autonomous Driving Applications: A Review,” *IEEE Trans. Intell. Transport. Syst.*, vol. 23, no. 1, pp. 33–47, 2022, doi: 10.1109/TITS.2020.3012034.

[3] A. Rudenko, L. Palmieri, M. Herman, K. M. Kitani, D. M. Gavrila, and K. O. Arras, “Human motion trajectory prediction: a survey,”

The International Journal of Robotics Research, vol. 39, no. 8, pp. 895–935, 2020, doi: 10.1177/0278364920917446.

[4] P. V. K. Borges, N. Conci, and A. Cavallaro, “Video-Based Human Behavior Understanding: A Survey,” *IEEE Trans. Circuits Syst. Video Technol.*, vol. 23, no. 11, pp. 1993–2008, 2013, doi: 10.1109/tcsvt.2013.2270402.

[5] D. Lee, C. Liu, Y.-W. Liao, and J. K. Hedrick, “Parallel Interacting Multiple Model-Based Human Motion Prediction for Motion Planning of Companion Robots,” *IEEE Trans. Automat. Sci. Eng.*, vol. 14, no. 1, pp. 52–61, 2017, doi: 10.1109/tase.2016.2623599.

[6] M. S. Shirazi and B. T. Morris, “Looking at Intersections: A Survey of Intersection Monitoring, Behavior and Safety Analysis of Recent Studies,” *IEEE Trans. Intell. Transport. Syst.*, vol. 18, no. 1, pp. 4–24, 2017, doi: 10.1109/tits.2016.2568920.

[7] A. Zyner, S. Worrall, and E. Nebot, “A Recurrent Neural Network Solution for Predicting Driver Intention at Unsignalized Intersections,” *IEEE Robot. Autom. Lett.*, vol. 3, no. 3, pp. 1759–1764, 2018, doi: 10.1109/lra.2018.2805314.

[8] J. Bock, R. Krajewski, T. Moers, S. Runde, L. Vater, and L. Eckstein, “The inD Dataset: A Drone Dataset of Naturalistic Road User Trajectories at German Intersections,” in *2020 IEEE Intelligent Vehicles Symposium (IV)*, 2020, doi: 10.1109/iv47402.2020.9304839.

[9] G. Aoude, J. Joseph, N. Roy, and J. How, “Mobile Agent Trajectory Prediction using Bayesian Nonparametric Reachability Trees,” in *Infotech@Aerospace 2011*, Reston, Virigina, 2011, doi: 10.2514/6.2011-1512.

[10] Chenhao Wang, Zhencheng Hu, and Roland Chapuis, “Predictive Lane Detection by Interaction with Digital Road Map,” *IMT*, vol. 7, no. 1, pp. 383–392, 2012. doi: 10.11185/imt.7.383. [Online]. Available: https://www.jstage.jst.go.jp/article/imt/7/1/7_1_383/_article/-char/ja/

[11] A. Amidi and S. Amidi. “Recurrent Neural Networks.” https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks (accessed Aug. 14, 2022).

[12] Q. Zou, H. Jiang, Q. Dai, Y. Yue, L. Chen, and Q. Wang, “Robust Lane Detection From Continuous Driving Scenes Using Deep Neural Networks,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 1, pp. 41–54, 2020, doi: 10.1109/tvt.2019.2949603.

[13] [4] https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Aug. 14, 2022).

[14] H. S. Hota, R. Handa, and A. K. Shrivas, “Time series data prediction using sliding window based RBF neural network,” *International Journal of Computational Intelligence Research*, vol. 13, no. 5, pp. 1145–1156, 2017. [Online]. Available: http://www.ripublication.com/ijcir17/ijcirv13n5_46.pdf

[15] H. W. Herwanto, A. N. Handayani, A. P. Wibawa, K. L. Chandrika, and K. Arai, “Comparison of Min-Max, Z-Score and Decimal Scaling Normalization for Zoning Feature Extraction on Javanese Character Recognition,” in *2021 7th International Conference on Electrical, Electronics and Information Engineering (ICEEIE)*, 2021, doi: 10.1109/iceeie52663.2021.9616665.

[16] Y. Xu and R. Goodacre, “On Splitting Training and Validation Set: A Comparative Study of Cross-Validation, Bootstrap and Systematic Sampling for Estimating the Generalization Performance of Supervised Learning,” *J. Anal. Test.*, early access. doi: 10.1007/s41664-018-0068-2.

[17] Q. Li, M. Yan, and J. Xu, “Optimizing Convolutional Neural Network Performance by Mitigating Underfitting and Overfitting,” in *2021 IEEE/ACIS 19th International Conference on Computer and Information Science (ICIS)*, 2021, doi: 10.1109/icis51600.2021.9516868.

[18] Y. Wang, Y. Li, Y. Song, and X. Rong, “The Influence of the Activation Function in a Convolution Neural Network Model of Facial Expression Recognition,” *Applied Sciences*, vol. 10, no. 5, p. 1897, 2020. doi: 10.3390/app10051897. [Online]. Available: https://www.mdpi.com/661772

[19] “neural networks activation function.” https://www.v7labs.com/blog/neural-networks-activation-functions