An End to End Learning based Ego Vehicle Speed Estimation System

Hitesh Linganna Bandari
Department of Electronics and Communication Engineering
Amrita School of Engineering, Coimbatore, India
Amrita Vishwa Vidyapeetham
cb.en.p2ael19006@cb.students.amrita.edu

Binoy B Nair
Department of Electronics and Communication Engineering
Amrita School of Engineering, Coimbatore, India
Amrita Vishwa Vidyapeetham
b_binoy@cb.amrita.edu

2021 IEEE International Power and Renewable Energy Conference (IPRECON) | 978-1-6654-0137-1/21/$31.00 ©2021 IEEE | DOI: 10.1109/IPRECON52453.2021.9640927

***Abstract*— Estimating speed of a vehicle from only the video captured from within the vehicle without relying on any other supplementary information such as GPS coordinates or LIDAR data, is a challenging task. A deep neural network-based technique is proposed in this work that uses only the recordings from onboard dash camera, to estimate speed of the ego car. Three models are designed using various combinations of LSTM and CNN and evaluated on two benchmark datasets. From the results, it is observed that the proposed systems are well capable of generating estimates of the ego vehicle speeds with good accuracy.**

Keywords*— *CNN-LSTM, End-to-End Learning, Ego vehicle speed, Speed Estimation, Sequence-to-Sequence Regression

I. INTRODUCTION

According to WHO statistics, approximately 1.35 million deaths are reported each year as a result of road traffic accidents [1]. In addition to the human cost, the economic impact is also severe, approximated to cost most countries around 3% of their gross domestic product.

Over speeding is a major cause of a large number of such accidents. Hence, it is useful for the law enforcement/insurance companies if the speed of the vehicle just before the crash could be ascertained. There are two common ways of doing this, namely: a) analysis of Black Box recordings and b) analysis of CCTV footage.

Black Box, which is also called as an Event Data Recorder (EDR) or an Accident Data Recorder (ADR) is an electronic device which is installed in-vehicle, its place of installation is such that there is no damage to the device even after a serious crash. This is cost inefficient compared to other available sources. Using the recordings from the CCTV (closed-circuit television) footage is cost efficient solution but the possibility of the CCTV camera being present in the vicinity of the crash are very less. Hence, in this study, a system based on deep-learning , that can estimate the ego vehicle speed based on the recordings from the dash cam is presented.

There are multiple methods which have been developed to estimate the ego vehicle speed, that make use of recent technologies. Han [2] proposed an approach to estimate the ego vehicle (car) speed or the other car's speed from the footages of the car-mounted camera collected from black box. Data is collected from recordings of 96 cases of respective black boxes and based on various factors classification is done for analysis like: travel circumstances and directions.

Chen et al. [3] presented a Driving Behavior Net (DBNet), the dataset that provides a 3D point cloud, drivers' behavior and recorded videos using dash cam. Extensive experiments illustrate that import of extra depth information to networks would help indeed to determine driving policies. Teed et al. [4] have introduced a new deep network architecture for optical flow, named as Recurrent All-Pairs Field Transforms (RAFT). This method gives high efficiency in parameter count, training speed and inference time , as well as strong cross-dataset generalization. Rill [5], worked on the estimation of the speed of self-driven vehicle on the KITTI benchmark making use of the DNN based optical flow & depth prediction techniques. In general, it can be seen from the literature that deep learning based solutions appear to be gaining popularity for vehicle related applications, e.g.[6] and [7].

Since time-dependence of video frames is an important factor while attempting to calculate speed from a sequence of such images, Long Short-Term Memory (LSTM) [8] models have been found to be ideally suited for the system proposed in this study. This is primarily due to the fact that the architecture of LSTM helps the designed model in remembering the information extracted from the previously fed inputs. LSTMs are being increasingly used in ADAS applications that need to consider time-dependence as a factor. LSTM based systems have been used for highway trajectory prediction as in[9], [10], and [11], prediction of road user/pedestrian trajectories as in [12] and [13] and for applications such as driver intention prediction, as in [14] and [15]. LSTM based models have also been used for such varied applications as detection of advanced persistent threats [16]. It was observed from the surveyed literature that there are very few datasets available which can be used to train and validate models for speed estimation applications. In this study, the designed models are validated on two datasets: (a) DBNet[3] and (b) a dataset used as a part of the Comma.AI speed challenge [17]. Our results are in line with these.

The remainder of the paper is organized as follows: The methodology is presented in section II, section III describes the results achieved and the section IV consists of the conclusions.

II. METHODOLOGY

Three different models are designed and evaluated in this study for estimating ego vehicle speed from dash cam images. This section presents the details of the deep learning based model architectures considered in this study.

A. Basic Structure of LSTM

The deep learning based models considered in this study are based on Convolutional Neural Networks (CNNs)[18] and LSTMs[8] . CNNs have been demonstrated to be capable of achieving very high accuracies for image classification applications, while the architecture of LSTM helps the designed model in remembering the information extracted from the previously fed inputs.

Architecture of LSTM and the flow of data can be seen visually in Fig 1. There are two connections from one block

978-1-6654-0137-1/21/$31.00 ©2021 IEEE

to another, the connecting line on the top is giving the cell information from previous time instant 'C_{t-1}' to present time instant block and the connecting line below feeds the hidden state of previous time instant 'h_{t-1}' to present time instant block. Input at the present time instant 'X_t' is concatenated with past hidden state 'h_{t-1}' before processing further. The relationships between different variables represented in Fig.1 is represented using equations (1)-(6) below[19]. In the equations (1)-(6) below, $\sigma(.)$ represents the sigmoid activation function.

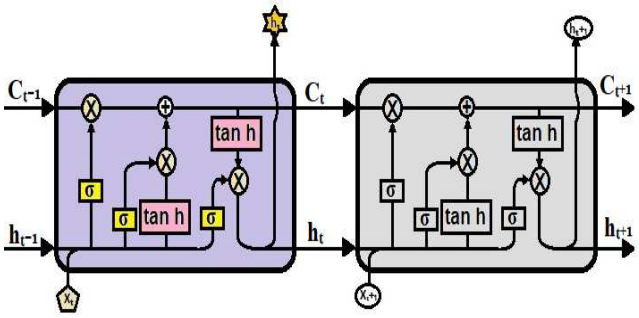


Fig.1 Basic LSTM block architecture (adapted from [19])

$$f_t = \sigma(W_f * [h_{t-1}, X_t] + b_f) \quad (1)$$

$$i_t = \sigma(W_i * [h_{t-1}, X_t] + b_i) \quad (2)$$

$$\tilde{C}_t = tanh(W_c * [h_{t-1}, X_t] + b_c) \quad (3)$$

$$O_t = \sigma(W_o * [h_{t-1}, X_t] + b_o) \quad (4)$$

$$h_t = O_t * tanh(C_t) \quad (5)$$

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t \quad (6)$$

From above equations we can understand how the cell state and hidden state are calculated. In a single LSTM cell, how the unwanted information is removed can be understood from (1), that is functionality of forget gate. Also useful information is extracted by making use of input 'X_t', preceding cell state 'C_{t-1}' and preceding hidden state output 'h_{t-1}' as shown in (6). From present cell state 'C_t', 'X_t' and 'h_{t-1}', present hidden state 'h_t' is calculated as shown in (5).

B. Basic Structure of CNN

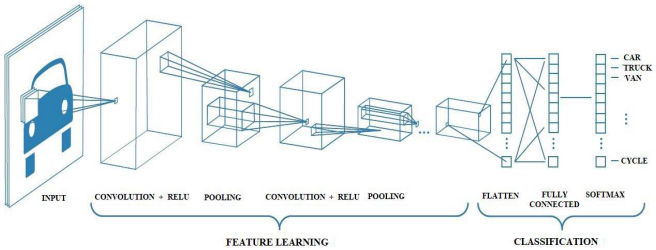


Fig. 2 Basic CNN architecture [20]

A CNN is typically made up of a sequence of layers, with the three main layers being convolution (or conv), pooling and fully-connected layers, usually, stacked together. The core building block of the CNN is the conv layer that does the most of computational work. There are three hyperparameters that are needed to define this layer, namely, (a) the depth (which is typically the number of filters needed), (b) the Stride (typically denoted by the variable 'S', this hyperparameter decides the steps in which the filter moves to cover the input dimensions, with the typical values being 1 or 2) and (c) the padding that is used to augment the size of the output (usually, a single or double zero padding is used, but in several cases there is no zero padding applied as well.).

A 'pooling' layer is typically added after the convolution layer to suppress noise as well as to reduce the amount of computations associated with the CNN. Typically, two different versions exist, namely, average pooling and max pooling. Max pooling is the more popular approach and is the only pooling technique now preferred.

Flatten layer is used to convert an image into a column vector which is used as input to the subsequent fully connected layer. All activations of the previous layer have full connections to the neurons in the fully connected layer.

ReLU (Rectified Linear Unit) is an activation function that simply generates its output '*Y*' for an input '*X*' based on the relationship given in (7):

$$Y = Max\ (0, X) \quad (7)$$

Dropout is a regularization technique that is usually used to mitigate the problem of overfitting. During training certain set of neurons are ignored at random. The batch normalisation layer, as the name suggests, normalizes the output of the previous layer. This is carried out by subtracting the mean value of the batch and dividing by the standard deviation of the batch. It replaces dropout, because of its regularizing effect. Softmax layer takes the total input to the *i*-th output neuron (y_i) and passes it through the softmax function $S(y_i)$. This results in the output of each neuron showing up as a probability value between 0 and 1.

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}} \quad (8)$$

C. Model Architectues

1. Model Architecture 1: Simple Convolution – LSTM

A basic CNN-LSTM network is built as shown in Fig. 3. Here the input images are fed to the network in sequence and these images are converted to an array of images using sequence folding so that convolution operation can be applied independently to each time step. The output from this layer is then given as input to convolution layer for segmentation and

TABLE I. LAYERS AND THEIR SPECIFICATIONS FOR MODEL 1

Node	Parameters
Input	Image input size: 66x66x3
Fold	Sequence folding layer, Batch Size: 10
Conv1	Filters: 16, Size: 5x5, Stride: 1, Padding: 0
Bn1	Batch Normalization
Relu1	ReLu Layer
Unfold	Sequence unfolding layer, Batch Size: 10
flatten	Flattening layer: 61504x1 Vector
Lstm1	LSTM Layer: Hidden Units 256
Fc	Fully Connected Layer, o/p: 1 Neuron
rl	Regression layer

classification of the images. The output from previous layer is given as input to sequence unfolding whose functionality is to restore the sequence structure and convert this array back to image sequences.

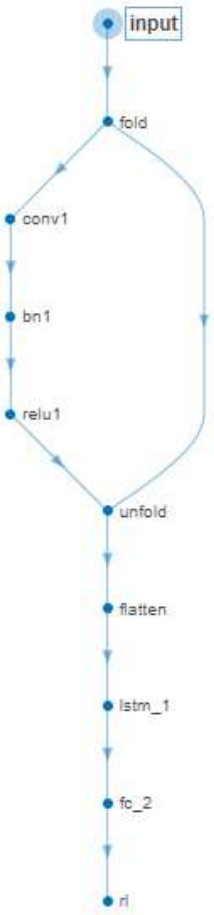


Fig. 3: Model Architecture 1- The simple convolution – LSTM Network architecture.

Now the sequence of images is fed to flatten layer which will convert image into feature vectors. Here the output of this feature vector is given as input to LSTM layer where the learning is done.

2. *Model Architecture 2:VGG 16–LSTM*

This architecture is based on the VGG-16[21] architecture. To this existing network, sequence folding, sequence unfolding, LSTM, Flatten and fully Connected layers were added for improving the performance. Their functionality is same as for model architecture 1. The network architecture for model 2 is as shown in Fig. 4. Here LSTM layer is used for sequence to sequence prediction of speed values which is our expected ego vehicle speed.

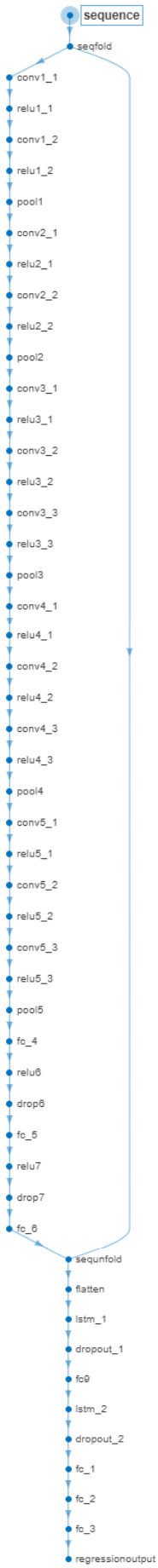


Fig. 4: Model Architecture 2-VGG 16 – LSTM Network.

3. *Model Architecture 3: Parallel Convolutional Layers – LSTM*

This model architecture is derived from the study presented in [22]. In this architecture, convolution layers are connected as shown in Fig. 5. Then it is connected to LSTM layer as shown, for speed prediction.

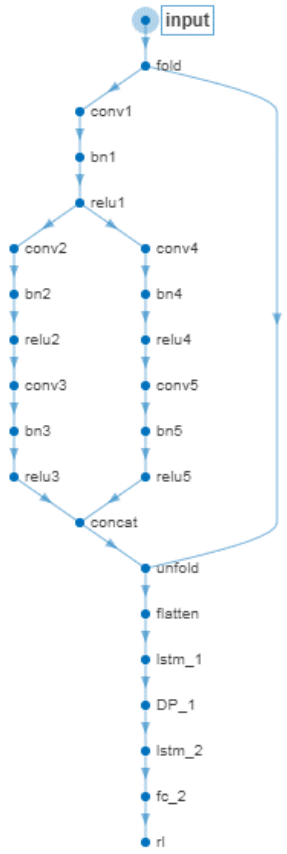


Fig. 5 Model Architecture 3-Parallel Convolutional Layers–LSTM Network.

TABLE II. LAYERS AND THEIR SPECIFICATIONS FOR MODEL 3

Node	Parameters
Input	Image input size: 66x66x3
Fold	Sequence folding layer, Batch Size: 10
Conv1	N: 64, D: 5x5, S: 1, P:0
BnX	Batch Normalization Layer
ReluX	ReLu Layer
Conv4	N: 32, D: 5x5, S: 1, P:0
Conv5	N: 16, D: 5x5, S: 1, P:0
Conv2	N: 32, D: 5x5, S: 1, P:0
Conv3	N: 16, D: 5x5, S: 1, P:0
Concat	Concatenation
Unfold	Sequence unfolding layer, Batch Size: 10
Flatten	Flattening layer: 93312x1 Vector
Lstm_1	LSTM Layer: Hidden units 256
DP	Dropout Layer, rate: 0.2
Lstm_2	LSTM Layer: Hidden units 128
Fc	Fully Connected Layer, o/p: 1 Neuron
Rl	Regression Layer

Note: In Table II, N: Number of filters, D: Filter dimensions, S: Stride, P: Padding.

Performance of each of these three models is presented in the next section.

III. EXPERIMENT RESULTS

A. *Datasets*

All the models considered in this study are validated on two datasets, namely DBNet[3] and the Comma.ai challenge dataset [17]. Comma.ai challenge dataset has speed resolution of 0.001 km/h. There are a total of 20,400 frames which is split as 90% for training and 10% for testing. The DBNet dataset consists of 11,520 frames which is divided into training, testing and validation sets. The dataset has images at two different resolutions, 1080 x 1920 and 66 x 200.

The dataset consists of per frame speed and steering angles with a speed resolution of 0.1km/h. The steering angle resolution per frame is also available, however, it has not been considered relevant for the purpose of speed estimation and has not been considered.

B. *Results*

As the preprocessing step, each frame is resized into 66x66x3. All the models are trained using the training parameters as follows:

TABLE III. LEARNING ALGORITHM PARAMETERS

Algorithm	SGD (Stochastic Gradient Descent)
Optimizer	Adam [23]
Gradient Decay Factor	0.9
Gradient Threshold	1
Max Epochs	10
Mini Batch size	10
Epsilon	10^{-8}
Initial Learning Rate	10^{-3}
Learning rate	0.001
Learning rate Drop Factor	0.1
L_2 Regularization	10^{-4}
Loss function	Cross Entropy

The loss function plots and the RMSE plots for the training phase are presented in Fig.6 -Fig.11. The model performance for the validation datasets is given in Table IV.

It is observed from the performance plots that the loss function and the training RMSE values remain stable over the given number of epochs considered in this study. It is also observed that the models perform much better in the case of comma.ai dataset rather than the DBNet dataset.

TABLE IV. MODEL PERFORMACE FOR THE VALIDATION DATASET

Model	Validation RMSE (Km/h)	
	DBNet	Comma.ai
Model 1 (Simple convolution – LSTM Network)	15	3.2
Model 2 (VGG16–LSTM Network.)	9	4.6
Model 3 (Parallel Convolutional Layers–LSTM)	12	3.2

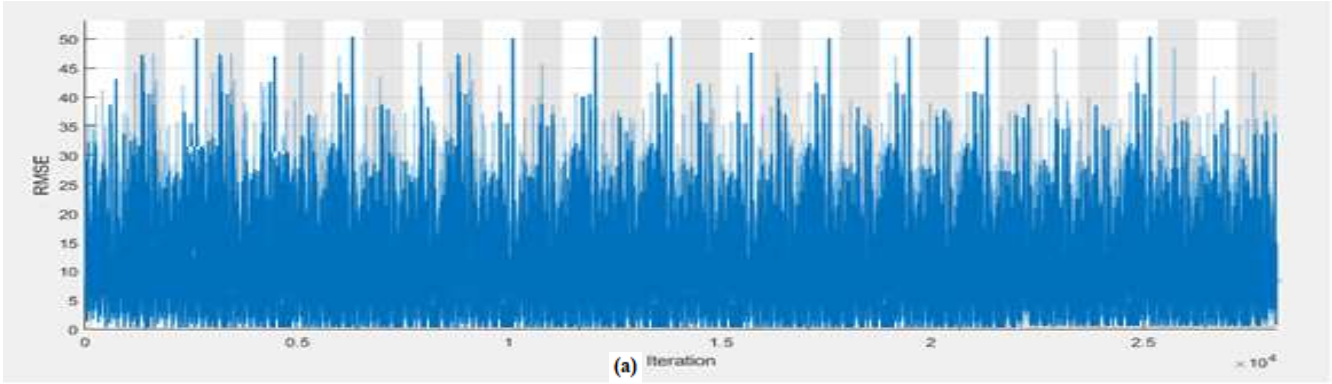


Fig. 6 a) RMSE for Model 1 (DBNet dataset) b) Loss for Model 1 (DBNet dataset)

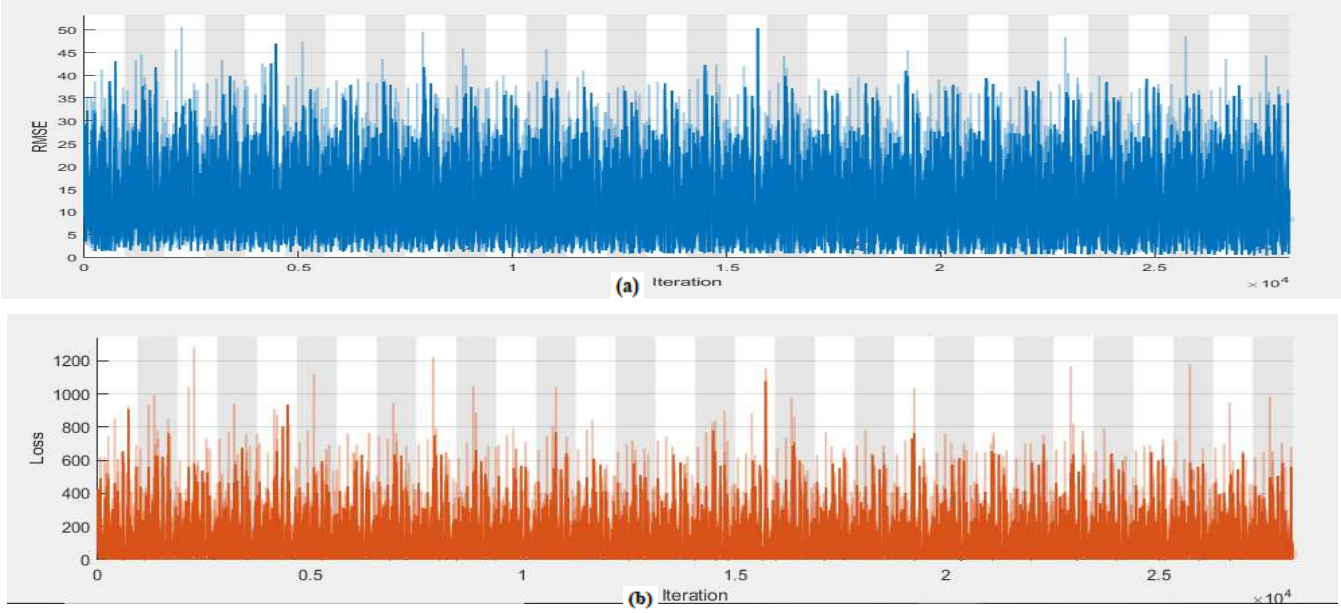
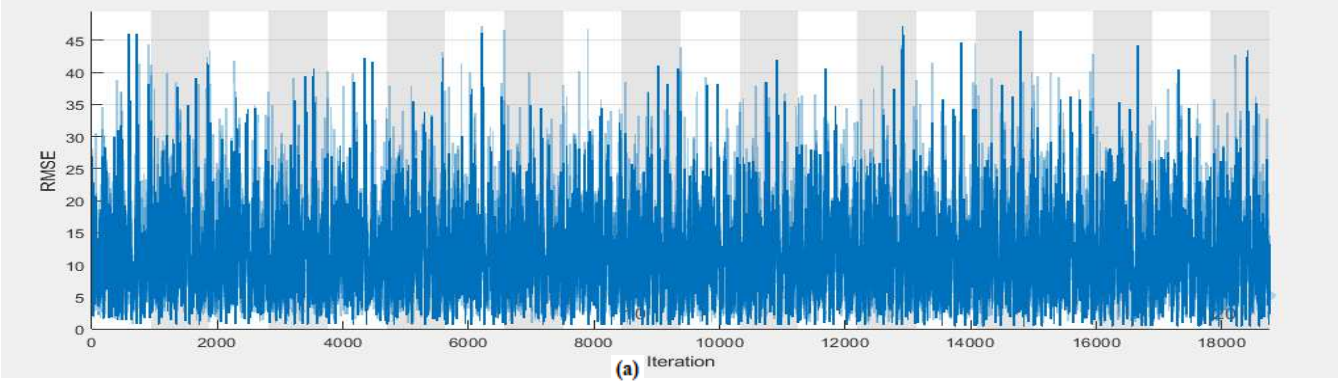


Fig. 7 a) RMSE for Model 2 (DBNet dataset) b) Loss for Model 2 (DBNet dataset)



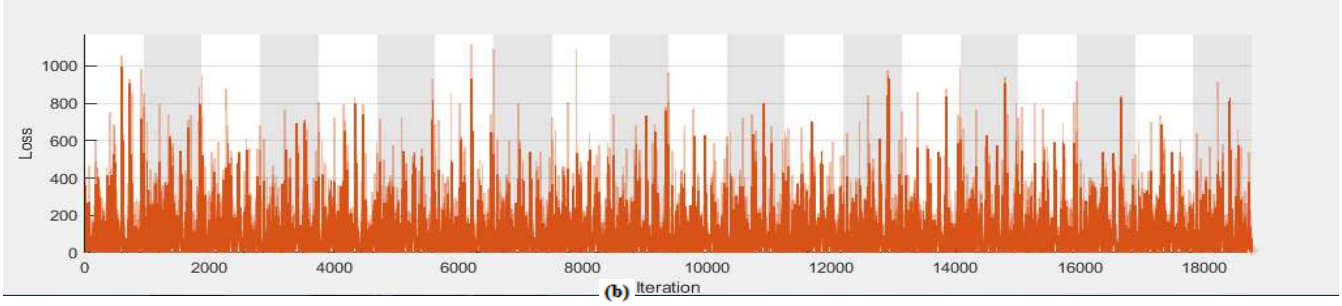


Fig. 8 a) RMSE for Model 3 (DBNet dataset) b) Loss for Model 3 (DBNet dataset)

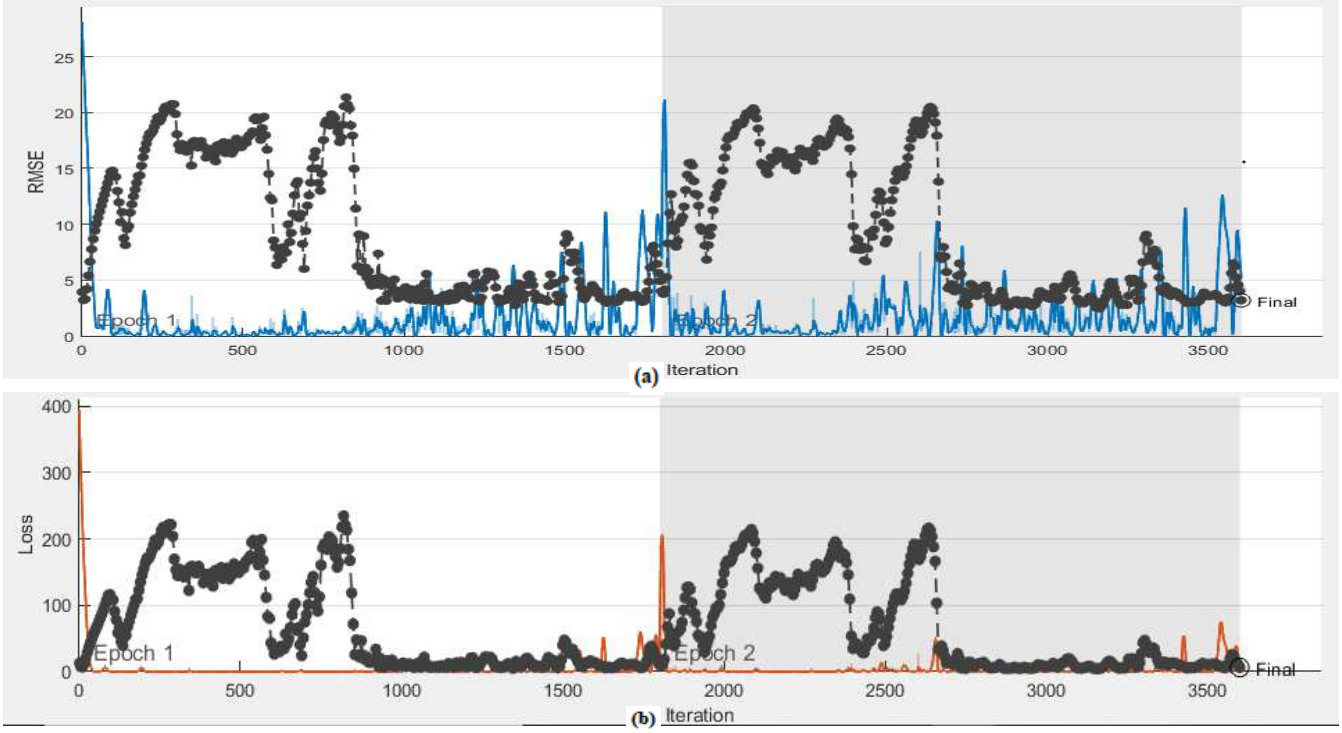


Fig. 9 a) RMSE for Model 1 (Comma.ai dataset) b) Loss for Model 1 (Comma.ai dataset)

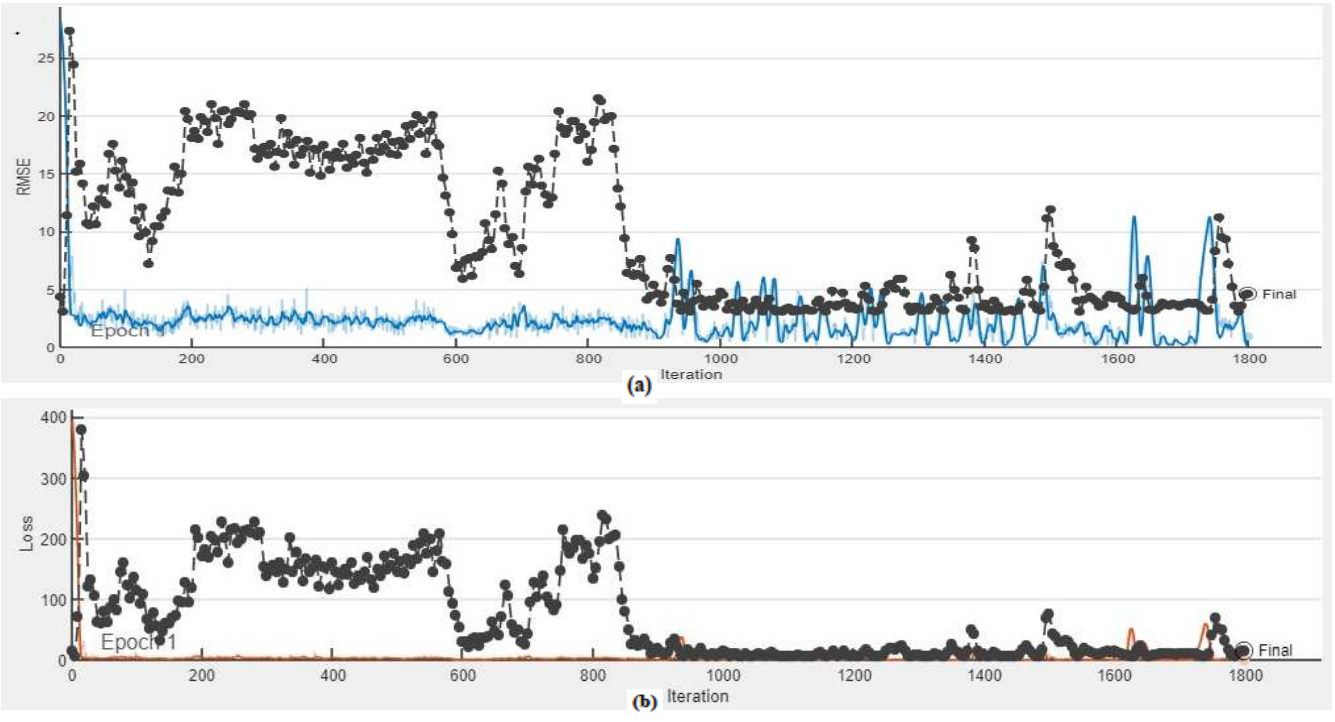


Fig. 10 a) RMSE for Model 2 (Comma.ai dataset) b) Loss for Model 2 (Comma.ai dataset)

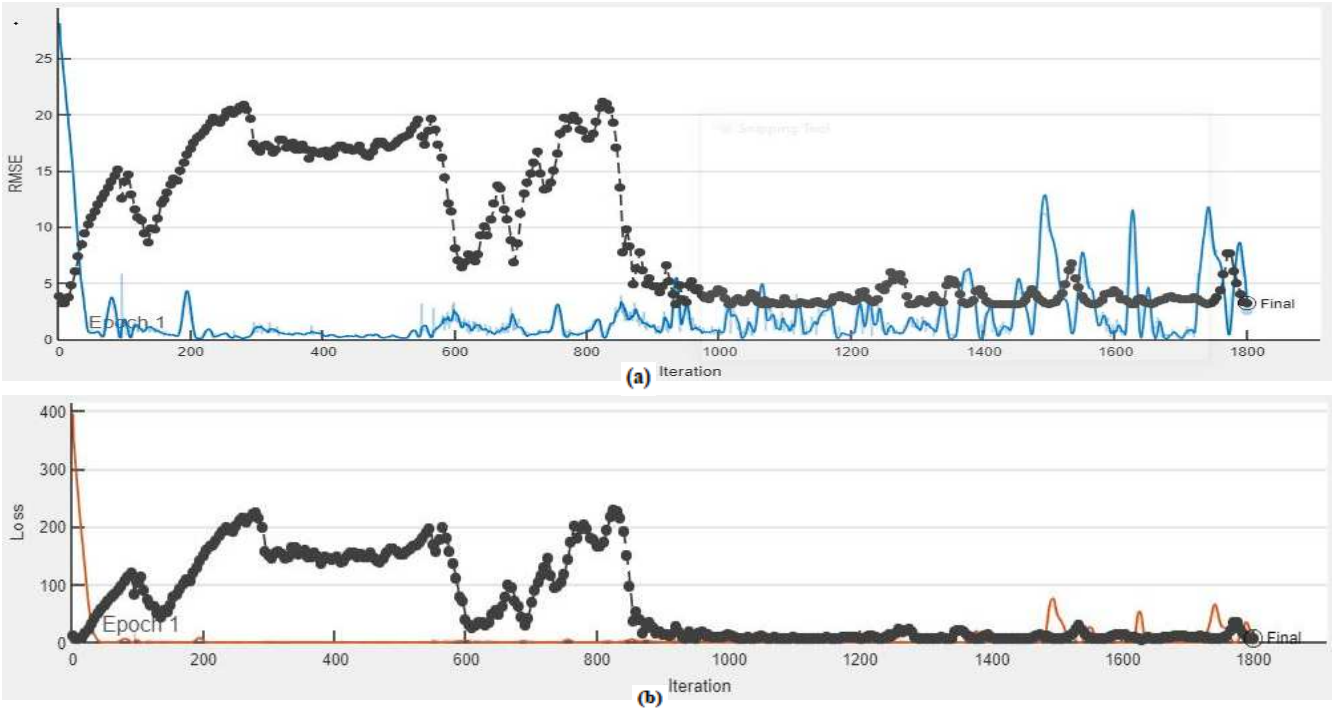


Fig. 11 a) RMSE for Model 3 (Comma.ai dataset) b) Loss for Model 3 (Comma.ai dataset)

IV. Conclusions

Three different deep-learning based models for estimation of speed from the dashcam footage have been designed and evaluated in this study. It is observed from the results that the three models are well capable of detecting speed of the ego vehicle from the dashcam video frames alone, without the need for any supplementary data such as GPS coordinates as evidenced from the low RMSE values of 3.2 km/h obtained for the comma.ai dataset and the reasonable RMSE of 9 km/h for the DBNet dataset. There is further scope for improvement in the performance, however, with the ultimate target being the validation RMSE of below 1 Km/h.

References

[1] Newsroom, "Road traffic injuries," *Fact sheets - World Health Organization*, 2020. https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries (accessed Apr. 12, 2021).

[2] I. Han, "Car speed estimation based on cross-ratio using video data of car-mounted camera (black box)," *Forensic Sci. Int.*, vol. 269, pp. 89–96, 2016, doi: 10.1016/j.forsciint.2016.11.014.

[3] Y. Chen *et al.*, "Lidar-video driving dataset: Learning driving policies effectively," in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5870–5878, [Online]. Available: http://www.dbehavior.net.

[4] Z. Teed and J. Deng, "RAFT: Recurrent All-Pairs Field Transforms for Optical Flow," in *European Conference on Computer Vision*, 2020, pp. 402–419, doi: 10.1007/978-3-030-58536-5_24.

[5] R. A. Rill, "Speed estimation evaluation on the KITTI benchmark based on motion and monocular depth information," 2019.

[6] R. Meena, B. B. Nair, and N. R. Sakthivel, "Machine Learning Approach to Condition Monitoring of an Automotive Radiator Cooling Fan System," in *Advances in Electrical and Computer Technologies*, 2020, pp. 1007–1020.

[7] Rahul and B. B. Nair, "Camera-based object detection, identification and distance estimation," in *2nd International Conference on Micro-Electronics and Telecommunication Engineering*, 2018, pp. 203–205, doi: 10.1109/ICMETE.2018.00052.

[8] S. Hochreiter and J. Schmidhuber, "Long Short-Term Memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997, doi: 10.1162/neco.1997.9.8.1735.

[9] F. Altché and A. de La Fortelle, "An LSTM network for highway trajectory prediction," in *2017 IEEE 20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 353–359.

[10] A. D. Kumar, R. Karthika, and K. P. Soman, "DeepTrackNet: Camera Based End to End Deep Learning Framework for Real Time Detection, Localization and Tracking for Autonomous Vehicles," in *International Conference on Intelligent Computing, Information and Control Systems*, 2019, pp. 299–307.

[11] S. H. Park, B. Kim, C. M. Kang, C. C. Chung, and J. W. Choi, "Sequence-to-sequence prediction of

vehicle trajectory via LSTM encoder-decoder architecture," in *2018 IEEE Intelligent Vehicles Symposium (IV)*, 2018, pp. 1672–1678.

[12] K. Saleh, M. Hossny, and S. Nahavandi, "Intent prediction of vulnerable road users from motion trajectories using stacked LSTM network," in *20th International Conference on Intelligent Transportation Systems (ITSC)*, 2017, pp. 327–332.

[13] H. Xue, D. Q. Huynh, and M. Reynolds, "A location-velocity-temporal attention LSTM model for pedestrian trajectory prediction," *IEEE Access*, vol. 8, pp. 44576–44589, 2020.

[14] N. Khairdoost, M. Shirpour, M. A. Bauer, and S. S. Beauchemin, "Real-Time Driver Maneuver Prediction Using LSTM," *IEEE Trans. Intell. Veh.*, vol. 5, no. 4, pp. 714–724, 2020.

[15] Z. Q. Liu, M. C. Peng, and Y. C. Sun, "Estimation of Driver Lane Change Intention Based on the LSTM and Dempster-Shafer Evidence Theory," *J. Adv. Transp.*, pp. 1–11, 2021, doi: 10.1155/2021/8858902.

[16] P. V. S. Charan, T. G. Kumar, and P. M. Anand, "Advance Persistent Threat Detection Using Long Short Term Memory (LSTM) Neural Networks," in *International Conference on Emerging Technologies in Computer Engineering*, 2019, pp. 45–54.

[17] Comma.ai, "Welcome to the comma.ai Programming Challenge!," *GitHub.com - comma.ai speed challenge*, 2018. https://github.com/commaai/speedchallenge (accessed Jan. 03, 2021).

[18] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Adv. Neural Inf. Process. Syst.*, pp. 1–9, 2012, doi: http://dx.doi.org/10.1016/j.protcy.2014.09.007.

[19] C. Olah, "Understanding LSTM Networks," *colah's blog*, 2015. https://colah.github.io/posts/2015-08-Understanding-LSTMs/ (accessed Jan. 03, 2021).

[20] MathWorks, "How CNNs Work," *Convolutional Neural Networks*, 2021. https://in.mathworks.com/discovery/convolutional-neural-network-matlab.html (accessed Apr. 21, 2021).

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015, pp. 1–14.

[22] N. Kawaguchi, J. Nozaki, T. Yoshida, K. Hiroi, T. Yonezawa, and K. Kaji, "End-to-end walking speed estimation method for smartphone PDR using DualCNN-LSTM," in *CEUR Workshop Proceedings*, 2019, vol. 2498, pp. 463–470.

[23] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015*, 2015, pp. 1–15.