Real-Time Vehicle Speed Estimation Using Optical Flow

Tigran Melkonian
melkonian.t@northeastern.edu
Northeastern University
Boston, MA
April 24, 2020

***Abstract* — Real-time Video-based vehicle speed estimation is a foundational component in constructing the perception, planning, and control architectures of autonomous driving systems. In this paper I present two monocular video-based vehicle speed estimation systems based on state of the art deep CNN architectures, Nvidia and FlowNetS. My goal is to train and evaluate the performance of both deep CNN architectures and prove that FlowNetS, a CNN specifically architected for learning optical flow, is better suited to learn monocular dashcam video feature motion and more accurately predict vehicle speed estimates when compared to the simpler, baseline Nvidia architecture. If this is the case, then the quality and complexity of a deep CNN responsible for learning optical flow may have a significant influence on vehicle speed prediction performance.**

1 Objectives and Significance

The human brain is the most powerful and complex organ in the human body, made up of more than 100 billion nerves, that communicate through trillions of synapses, all necessary to construct an accurate, yet subjective, perception of information that is constantly changing in space and time within the world around us. We gather information about our world through a set of sensory organs, chief of which is the set of light-signal based information collected by our eyes. Creating an accurate representation of reality is crucial for human existence and survival and this is especially the case when operating vehicles, at speed, when one has to make crucial decisions within a fraction of a second. There are currently companies like Tesla, whose perception systems primarily rely on computer vision, that aim to reduce the baseline rate of human perception error behind the wheel through autonomous driving. A small, yet important, part required to solve autonomous driving, that specifically aids the development of robust planning systems, is to accurately and continuously estimate the speed of a car using a deep CNN.

The ultimate goal of this paper is to first understand, in general, how well deep neural networks perform at estimating the absolute speed of a car in real time when given a monocular dashboard video stream as input and whether CNN architectures specifically constructed for learning optical flow, such as FlowNetS, perform better than the baseline Nvidia architecture.

2 Background

2.1 Convolutional Neural Networks

Y. LeCun *et al.*[1] introduced the first implementation of the modern convolutional neural network in a 1998 study inspired by the functionality of the brain's visual cortex for the problem of learning via sparse data. Y. Lecun and his team continued to advance the founding work done from the 1950's - 1980's [11][12][13][14] by constructing a CNN architecture, called LeNet, which used back-propagation to automatically learn convolutional kernel coefficients from images of hand-written digits. Ultimately, demonstrating better generalizable performance when compared to the prior method of manually defining kernel coefficients.

Convolutional neural networks (CNNs) are a type of feed-forward fully connected input-output deep learning network that trains and tests inputs through a series of convolutional layers to, most commonly, analyze and

classify visual imagery. Although image analysis is the primary application, CNNs can also be used for other data analyses or classification problems such as recommender systems, natural language processing, and financial time series.

CNNs' architecture consists of three primary layers: First, the foundational building block, a convolutional operation applies a kernel to inputs from a window (subset) of the initial input, applies a bias, and will result in an activation with a local nonlinearity to ultimately create a feature map that summarizes features of interest in the input. After every convolution operation, a Nonlinear operation called Rectified Linear Unit (ReLu) equates input values that are less than zero to zero and input values greater than zero remain unchanged. Finally, the pooling operation is used to reduce the dimensionality of the input layers after any arbitrary layer the input has passed through within the CNN. Max pooling is a common technique used to reduce input dimensionality and preserve spatial invariance by first defining an operation window size, sliding that window across the input, and taking the maximum value within the window. Ultimately this architecture allows the network to concentrate on small low-level features that exist in the initial set of hidden layers, then eventually mapping to higher-level features in subsequent hidden layers. The figure below provides an overhead view of this neural architecture.

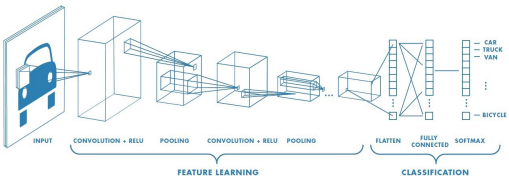

Figure 1: CNN architecture for image classification

2.2 Optical Flow Estimation: FlowNetSimple (FlowNetS)

Very broadly, Optical Flow is about globally understanding how objects and surfaces are moving between, at least two, consecutive images at the level of a pixel across the images or video stream. Optical flow has many applications such as image stabilization, video compression, object detection and tracking, navigation, and motion estimation.

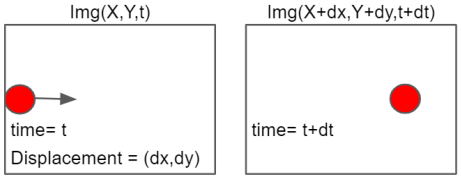

Figure 2: Estimating optic flow vector for image feature

Flow estimates are calculated between consecutive frames, where the image intensity (*Img*) is a function of pixel space (X,Y) with respect to time (t). For example if I take the colored object in image *Img(*X,Y,t) and move it by (dx,dy) pixels over time (dt), I obtain the new image *Img*(X+dx,Ydy,t+dt).

FlowNetS architecture uses nine convolutional layers with a nonlinear ReLu operation after each layer to calculate flow estimates from an input of two stacked sequential images to extract the motion information. This architecture does not have fully connected layers, so it cannot take input images of arbitrary size. The convolution filter decreases with depth of convolution at a stride of 2, so the first layer is seven by seven, the next two is five by five and the rest are three by three. An illustration of the FlowNetSimple architecture is provided below in Fig. 3.

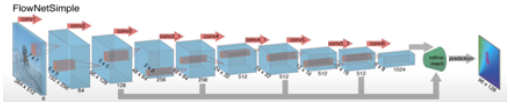

Figure 3: FlowNetS from P.Fischer *et al.* [4]

3 Proposed Approach

3.1 Data

Using a CNN to compute optical flow requires a large amount of training data. Data is currently the primary performance bottleneck because it is particularly difficult to obtain accurately labeled high quality video footage data. I will use the training data set provided by comma.ai for their "Speed Challenge" [10]. Their data set contains one video for training and a text file containing ground truth, per video frame, speed estimates. *train.mp4* is a 17-minute video of driving captured at 20 frames per seconds (fps) and contains 20400 frames. *train.txt* contains 20400 ground truth speed values of the vehicle that correspond to each frame in *train.mp4*.

I will split this dataset into dedicated training and validation sets that follow a 85/15 percent train/ validation split. In other words, I will use approximately 17340 images (8670 image pairs) for training and approximately 3060 images (1530) image pairs for validation. All sequenced image pairs are randomly selected without replacement to ensure a pure validation set.

Prior to being fed into any model, each image pair is preprocessed by constraining the input dimensionality to ignore static features that will not contribute meaningful flow information, i.e. the dashboard at the bottom of the video frame and the hood of the car at the top of the video frame. Additionally I augment the brightness of each image by multiplying the saturation by a uniform random variable to reduce glare from bright lights or the sun shining into the camera, which could negatively impact the optical flow calculation when generating the training vectors.

3.2 Method and Implementation

I constructed both the Nvidia and FlowNetS architectures using Python, Jupyter Notebook, and associated, state-of-the-art machine learning libraries such as TensorFlow 2.0 and Keras (as the backend). In terms of hardware for training the models, I had access to a system that contained an Nvidia GTX 1050 TI Max Q design GPU, Intel Core i7 CPU, 16 GB Ram, and 4 GB dedicated GPU memory. When training, given the dedicated training and validation splits constructed from *train.mp4* and *train.txt*, I set the hyper parameters to the following:

- **training batch size**: 32/64
- **validation batch size**: 32/64
- **number of epochs**: 25
- **training steps per epoch**: 536/268
- **validation steps per epoch**: 95/47

In summary I train and validate using the entire training and validation set per epoch.

I implemented an Adam Optimizer within each optic flow model to monitor for and minimize the observed mean square error between predicted speed estimates and ground truth speed estimates through each training step and epoch. Finally, I employed an early stopping method to prevent overfitting of the training data by setting the minimum acceptable decrease in loss of mean squared error between subsequent epochs to a value of .23 with a patience tolerance of three epochs, meaning if the loss does not meet the minimum requirement within three epochs I stop training.

For those interested in viewing my full implementation of all models and the training and development environment, please feel free to access my code shared in this GitHub repository: https://github.com/TigranMelkonian/MA7243

3.3 Evaluation Strategy

I will use Mean Squared Error (MSE), the average of squared errors. I aim to both understand the performance of each model, FlowNetS and Nvidia, with respect to the accuracy of speed estimations using dashcam data from Comma Ai and attempt to prove that FlowNetS will produce superior results when compared to the simpler, baseline Nvidia architecture. So, I would like to see lower MSE values for experiments that take into consideration more complex architectures specifically designed for learning feature motion in videos or continuous image sequences.

For each unique experiment, I will use the dashcam video from train.mp4 as input to train the model, then use test.mp4 and appropriately partitioned subset of train.txt, that will contain ground truth speed values for images in test.mp4 to test the performance of the newly trained model.

4 Results

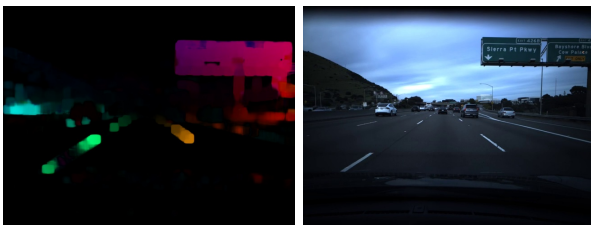

Figure 4: Detecting optic flow from sequential image inputs

4.1 Nvidia (Baseline)

I implemented the Nvidia CNN architecture [15] to generate baseline results in order to offer a performance comparison point for the FlowNetS model. Note from figure 5 below that after cycling through 8 epochs I was able to define a set of weights that achieved a minimum training validation MSE of 2.26.

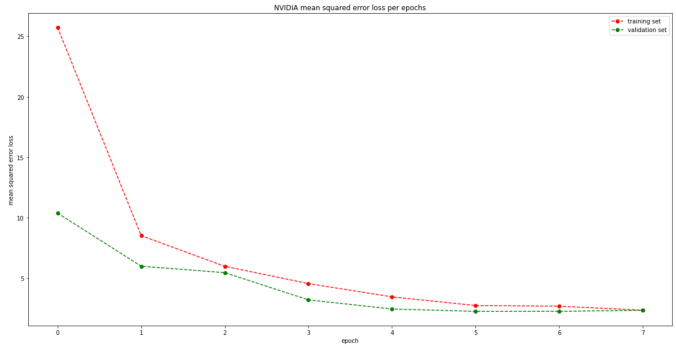

Figure 5: Mean squared error loss pre epoch - Nvidia model
Validation MSE: 2.26

Training loss for the baseline Nvidia model experiences a large decline of approximately 17 points in mse from the first epoc, at an mse of approximately 25.73, to the second epoch, at an mse of approximately 8.51. Thereafter, training loss steadily declines to a minimum mse of 2.36 at the eighth epoch. Validation loss starts at 10.4 at the first epoch and generally steadily declines with training loss. Validation loss reaches a minimum mse of 2.26 at the sixth epoch and unfortunately does not improve thereafter. I take the updated weights of the sixth epoch and stop training early (using a training tolerance of 3 epochs).

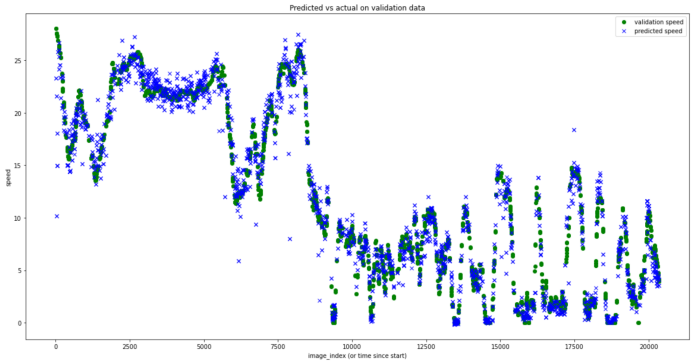

Figure 6: Observed (go) v.s. Predicted validation speed (bo)

From figure 8 above, it is easy to see that the baseline model generally does a great job at predicting the ground truth speed of the dedicated validation set. I can note that the model is able to predict large accelerations or decelerations in speed within short periods of time with greater accuracy than it is able to predict constant speeds or small accelerations or decelerations in speeds over longer periods of time.

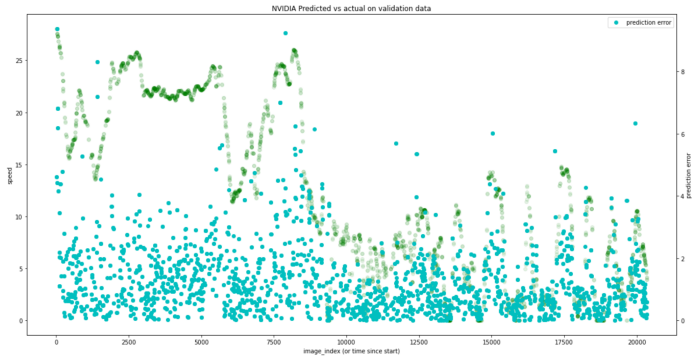

Figure 7: Predicted validation speed error (teal)

The minimum training validation mse of 2.26 equates to an average speed estimation error of 1.50 miles-per-hour (mph). Seemingly impressive, but we can do much better!

4.2 FlowNetS

Note from figure 8 below that after cycling through ten epochs I was able to define a set of weights that achieved a minimum validation MSE of 1.15. In comparison to the baseline Nvidia baseline model, FlowNetS was able to reduce the validation MSE by approximately 49%!

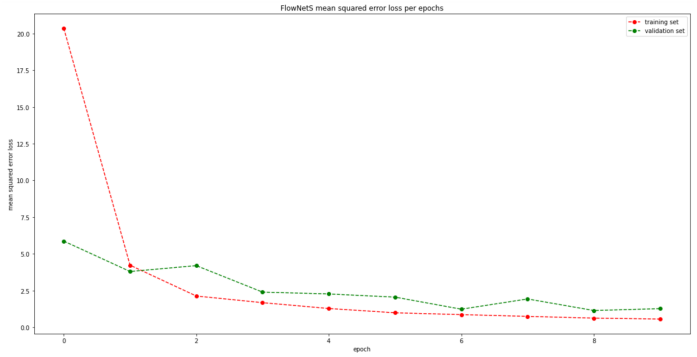

Figure 8: Mean squared error loss pre epoch - FlowNetS model
Validation MSE: 1.15

Training loss for the FlowNetS model experiences a large decline of approximately 16 points in mse from the first epoc, at an mse of approximately 20.35, to the

second epoch, at an mse of approximately 4.22. Thereafter, training loss steadily declines to a minimum mse of .56 at the tenth epoch. Validation loss starts at about 5.87 at the first epoch and steadily declines with training loss. On average validation loss is slightly greater than the training loss signaling a fairly good fit to the training data. Validation loss reaches a minimum mse of 1.15 at the ninth epoch and unfortunately does not improve thereafter. I take the updated weights of the ninth epoch and stop training early because at this point the training loss is at an mse of less than the minimum requirement to proceed to the next epoch.

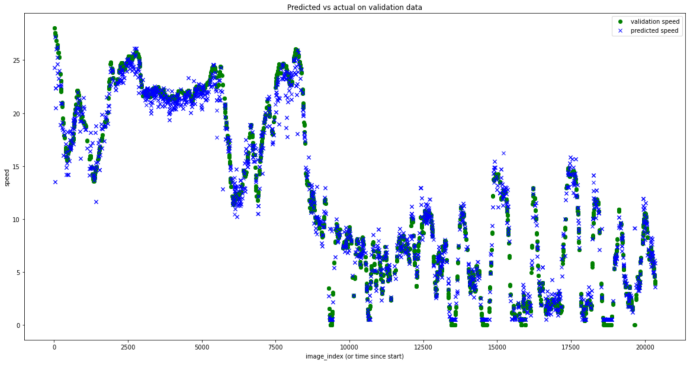

Figure 9: Observed (-go) v.s. Predicted validation speed (-bo)

From figure 9 above, it is easy to see that the FlowNetS model does a great job at predicting the ground truth speed of the dedicated validation set. I can note that the model is able to predict both large accelerations or decelerations in speed within short periods of time with greater accuracy and constant speeds or small accelerations or decelerations in speeds over longer periods of time. However, FlowNetS generally overestimates local maximum and minimum speed values after periods of acceleration or deceleration respectively. Overall the predictions of the FlowNetS model, in comparison to the baseline Nvidia model, seem to be more smoothed at high and low troughs of constant speed and more closely match speeds throughout short or abrupt periods of acceleration and deceleration.

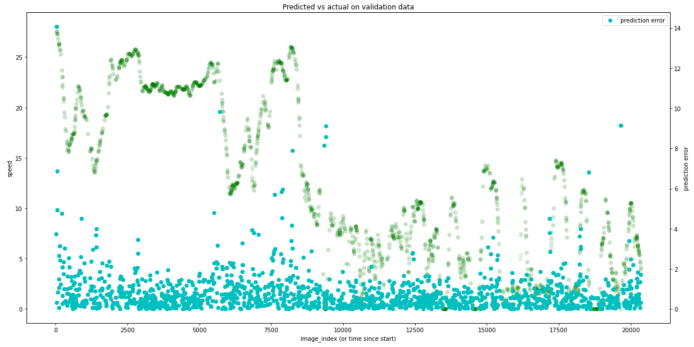

Figure 10: Predicted validation speed error (teal)

The minimum validation mse of 1.15 equates to an average speed estimation error of about 1.07 miles-per-hour (mph)!

Model	MSE (Train)	MSE (Validation)
Nvidia (baseline)	2.36	2.26
FlowNetS	.56	1.15

Figure 11: Table of training and validation MSE for Nvidia and FlowNetS

5 Conclusion

In this paper I initially set out to construct a system of state of the art deep CNN models for estimating a vehicle's speed via monocular dashcam video as input. I first reconstructed the baseline Nvidia model to reproduce speed estimation results previously accomplished by members of the SpeedChallenge community.

With the baseline Nvidia model, I was able to produce better results at an mse of 2.26 after heavy edits to the data preprocessing methodology I used as a guide in constructing the development environment for all training implementations. Next, I constructed FlowNetS according to the architecture shown in figure 3 without the refinement network at the end. I excluded the refinement network because it is not necessary for accurately estimating large movements and does not have a significant impact on results, however does reduce the complexity of the architecture altogether and consequently reduces the time necessary to train the model. With FlowNetS I was able to outperform the

performance of the baseline Nvidia model at an mse of 1.15. Thus reinforcing the idea that models architected specifically for learning optical flow between sequenced image pairs, with a monocular video stream as input, better learn the task of predicting vehicle speed from dashcam footage.

For future work, I would like to explore the performance of more complex flownet architectures, namely FlowNetCor and FlowNet2 (FlowNetCSS), on learning the task of estimating vehicle speed via monocular dashcam video stream. I suspect that more complex architectures can undoubtedly outperform the simplest flownet architecture, FlowNetS, however I believe the added complexity will significantly impact the training process length due to an increase in the number of necessary parameters that must be trained. Additionally, I would like to expand on the available data for this project by producing my own training data. Ultimately, with enough data, even the Nvidia (baseline) model would perform as good, if not better, than the FlowNetS model I have trained using the data available from comma.ai.

6 References

[1] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, 1989. Backpropagation applied to handwritten zip code recognition, Neural Comput. 1, 4 (December 1989), 541–551. DOI:https://doi.org/10.1162/neco.1989.1.4.541

[2] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox. FlowNet 2.0: Evolution of optical flow estimation with deep networks. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 1647–1655, 2017.

[3] Fischer, Philipp, Dosovitskiy, Alexey, Ilg, Eddy, Hausser, Philip, Hazrba, Caner, Golkov, Vladimir, van der ¨ Smagt, Patrick, Cremers, Daniel, and Brox, Thomas. Flownet: Learning optical flow with convolutional neural networks. In ICCV, 2015.

[4] K. Kale, S. Pawar, and P. Dhulekar. Moving object tracking using optical flow and motion vector estimation. In Reliability, Infocom Technologies and Optimization (ICRITO)(Trends and Future Directions), 2015 4th International Conference on, pages 1–6. IEEE, 2015.

[5] D. Zhang, H. Maei, X. Wang, and Y.-F. Wang. Deep reinforcement learning for visual object tracking in videos. arXiv preprint arXiv:1701.08936, 2017.

[6] S. Hua, M. Kapoor and D. C. Anastasiu, "Vehicle Tracking and Speed Estimation from Traffic Videos," 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW), Salt Lake City, UT, 2018, pp. 153-1537, doi: 10.1109/CVPRW.2018.00028.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In NIPS, pages 1106–1114, 2012.

[8] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Deep end2end voxel2voxel prediction (the 3rd workshop on deep learning in computer vision). In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2016.

[9] comma.ai. (2018, July 11). *SpeedChallenge*. https://github.com/commaai/speedchallenge/tree/master/data

[10] Fukushima K. Neocognitron: a self organizing neural network model for a mechanism of pattern recognition unaffected by shift in position.

[11] Ciresan, Dan; Meier, Ueli; Schmidhuber, Jürgen (June 2012). Multi-column deep neural networks for image classification. 2012 IEEE Conference on Computer Vision and Pattern Recognition. New York, NY: Institute of Electrical and Electronics Engineers (IEEE). pp. 3642–3649.

[12] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". Nature. 521 (7553): 436–444.

[13] Hubel, D. H.; Wiesel, T. N. (1968-03-01). "Receptive fields and functional architecture of monkey striate cortex". The Journal of Physiology. 195 (1): 215–243.

[14] https://developer.nvidia.com/blog/deep-learning-self-driving-cars/

[15] P.-H. Huang, K. Matzen, J. Kopf, N. Ahuja, and J.-B. Huang. Deepmvs: Learning multi-view stereopsis. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 2821–2830, 2018.