Real-Time Vehicle Speed Estimation Using Optical Flow and MonoDepth

Tigran Melkonian, Kuhoo Saxena
melkonian.t, saxena.k @northeastern.edu
Northeastern University
Boston, MA
December 15, 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 we present a monocular video-based vehicle speed estimation system based on a proposed deep CNN architecture that considers optical flow and depth disparity as input information. We aim to train and evaluate the performance of our proposed deep CNN, optical flow based, architectures and prove that combining depth disparity information with optical flow vectors can generally improve predicted vehicle speed mean squared errors and accuracies when compared to using optical flow alone and the quality and complexity of a deep CNN responsible for learning optical flow can have a significant influence on vehicle speed prediction performance. Our best optical flow architecture in combination achieved a mse of 1.25 for predicted vehicle speed against a dedicated validation set containing ground truth speed values.**

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. A perfectly functioning set of human eyes can process ten to twelve images a second, differentiate between approximately ten million colors, perceive three dimensional environmental depth, and yet they are still subject to the shortcomings attributable to human error such as distortion of information, information loss, and creation of mis-information, that can all lead to the construction of a poor representation of reality. 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 understand how well a deep neural network performs at estimating the absolute speed of a car in real time given a monocular dashboard video stream. We will initially implement two state-of-the-art optical flow networks, FlowNet, FlowNetCorr, and FlowNet2-CSS to generate

separate baseline speed estimation values on a dataset, provided by comma.ai for their "Speed Challenge". Next, we aim to improve on the baseline accuracy of each FlowNet architectures by producing a scaled speed estimate that will combine flow vectors with monocular depth disparity. We will rely on the intuition that the magnitude of flow estimates is directly correlated with the moving speed of the camera and that features within an image that are closer to the camera seem to move faster than those more distant. We believe that a scaled speed score derived from the combination of flow estimates and monocular depth information will improve the absolute speed estimation accuracy.

2 Background

2.1 Deep Learning

Deep Learning's goals are to extract useful patterns from data in an automated fashion through constructing and optimizing neural networks. At the core, deep learning provides the ability to form high levels of representation of data necessary to understand and interpret the data. With respect to most learning algorithms, deep learning has a proven record of increasing performance metrics with larger datasets by automating the extraction of meaningful features. Deep learning architectures such as deep neural networks and convolutional neural networks have been applied to fields including computer vision and natural language processing. Recent advancements in the computation ability of personal computers have made the task of implementing and training deep learning models possible. Work done within the deep learning community such as A. Krizhevsky *et al.* [8], who showed promising performance on large-scale image classification trained CNNs with backpropagation. E. Ilg *et al.* [3] proposed an end-to-end optical flow estimation model called FlowNet that produces a vectorized flow field for a sequence of two images. Since the work done by E. Ilg *et al.* [3], there have been many papers that iterated on the FlowNet model using CNNs: including a coarse-to-fine pyramidal approach, a 3D CNN D. Tran *et al.* [9], and a stacked approach that combines both "simple" and "corr" methods in FlowNet by E. Ilg *et al.* [3] through their creation of FlowNet2 which was proven to better manage producing flow estimates on large displacements of features of interest in subsequent images.

So far, only FlowNet2 has significantly outperformed FlowNet. Despite impressive compute speeds and quality of flow estimates, neither FlowNet nor FlowNet2 take into consideration single-view depth information. This oversight can lead to a degradation in performance when producing flow estimates for larger feature displacements. With this in mind, we aim to both understand the baseline performance of each model (FlowNetSimple, FlowNetCorr and FlowNet 2.0) with respect to the accuracy of speed estimations using dashcam data from Comma Ai and attempt to prove that a scaled speed that takes into consideration single-view depth information will produce superior results when compared to any one of the models individually.

2.2 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.

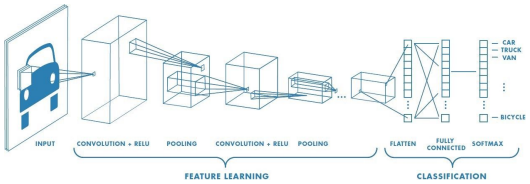

Figure 1: CNN architecture for image classification

2.3 Optical Flow Estimation: FlowNetSimple, FlowNetCorr, FlowNet2CSS

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.

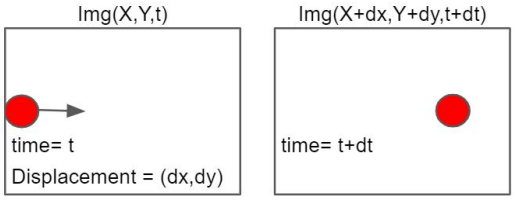

Figure 2: Estimating optic flow vector for image feature

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 we take the colored object in image *Img(*X,Y,t) and move it by (dx,dy) pixels over time (dt), we obtain the new image *Img*(X+dx,Ydy,t+dt).

FlowNetSimple 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. 2.

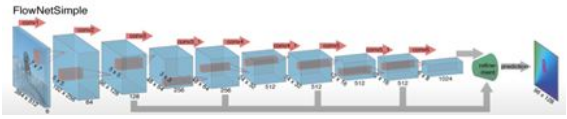

Figure 3: FlowNetS from P.Fischer *et al.* [4]

FlowNetCorr architecture attempts to facilitate the matching process that exists in FlowNetSimple by calculating flow estimates through two separate, yet identical processing streams (similar to Siamese). As described in P. Fischer *et al.* [4], the architecture first extracts features of two sequential images independently and then matches them based on correlative similarity. Given the feature maps of two sequential images f_1 and f_2 $R^2 \in R^c$, where w, h, and c define image width, height, and number of pixel channels, the initial correlation layer FlowNetCorr's architecture compares corelative similarity of each patch from f_1 with each patch from f_2. The correlation of two patches, located at x_1 in the first image and x_2 in the second is given:

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle$$

To minimize computational cost in generating correlation values, the maximum displacement for comparisons is limited to only include patches that exist in a neighborhood of 2d+1 where d is the maximum displacement. An illustration of the FlowNetCorr architecture is provided below in Fig.3.

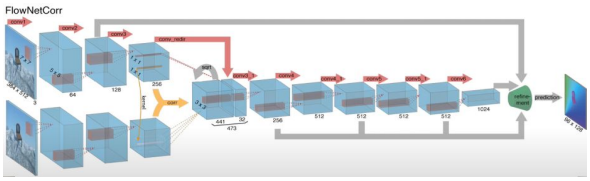

Figure 4: FlowNetCorr from P.Fischer *et al.* [4]

FlowNet2 Architecture is well equipped to compute large displacement flow estimates through integrating multiple FlowNets of different types (FlowNetSimple and FlowNetCorr). FlowNet2-CSS consists of stacking one FlowNetCorr and two back-to-back FlowNetSimple. E. Ilg *et al.* [3] has shown flow estimation performance improvements by approximately fifty percent over the original FlowNetCorr model. An illustration of FlowNet2-CSS is provided below in Fig.4 where braces indicate concatenation of inputs and Brightness Error is the difference between the first image and the second image warped with the previously estimated flow.

FlowNet2 Architecture is well equipped to compute large displacement flow estimates through integrating multiple FlowNets of different types (FlowNetSimple and FlowNetCorr). FlowNet2-CSS consists of stacking one FlowNetCorr and two back-to-back FlowNetSimple. E. Ilg *et al.* [3] has shown flow estimation performance improvements by approximately fifty percent over the original FlowNetCorr model. An illustration of FlowNet2-CSS is provided below in Fig.4 where braces indicate concatenation of inputs and Brightness Error is the difference between the first image and the second image warped with the previously estimated flow.

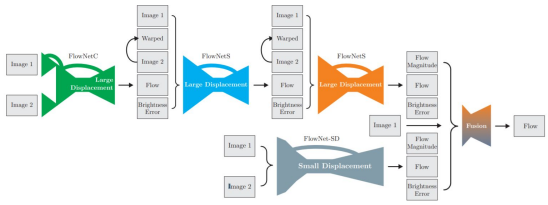

Figure 5: FlowNet2-CSS from E. Ilg *et al.* [3]

2.5 Monocular Depth Estimation: MonoDepth

Depth estimation from 2D images is a fundamental task in many applications like scene understanding and reconstruction. Having a dense depth map of the real-world can be very useful in applications including vehicle navigation and scene understanding, augmented reality, and

image reconstruction and segmentation. Recent developments in depth estimation are focused on using convolutional neural networks (CNNs) to perform 2D to 3D reconstruction. Current applications in augmented reality, synthetic depth-of-field, and other image effects require fast computation of high resolution 3D reconstructions in order to be applicable. For such applications, it is critical to faithfully reconstruct discontinuity in the depth maps and avoid the large perturbations that are often present in depth estimations computed using current CNNs.

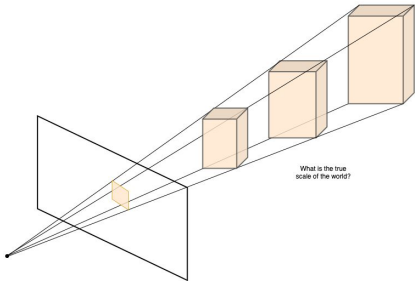

Figure 6: Depth Estimation

Most of the traditional methods for depth estimation rely on the assumption of having observations of the scene, either in space or time (e.g., stereo or multi-view, structure from motion). Traditional methods can be categorized in two sets, active and passive methods. Active methods involve computing the depth in the scene by interacting with the objects and the environment. In the category of passive methods, there are two primary approaches: (a) multi-view depth estimation, such as depth from stereo, and (b) monocular depth estimation.

Monocular depth cues is the set of information that allows humans to perceive depth and distance from a single image. Monocular Depth estimation is the deep learning task of obtaining a representation of spatial structure of a scene by determining the distance of features from a single image source of a dynamic scene. C. Godard *et al.* [2] constructed a convolutional neural network that innovates on existing learning based monocular image source depth estimation methods that produces disparity values used to construct a global dense depth map from two consecutive images.It is a fundamental challenge in computer vision and has potential applications in robotics, scene understanding, 3D reconstruction and medical imaging. This problem remains challenging as there are no reliable cues for perceiving depth from a single image. For example, temporal information and stereo correspondences are missing from such images. The classical depth estimation approaches heavily rely on multi-view geometry such as stereo image. These methods require alignment and calibration procedures which are important for multi-camera or multi-sensor depth measurement systems. Multi-view methods acquire depth information by utilising visual cues and different camera parameters. Most of the binocular or multi-view methods are able to estimate fairly accurate depth information. However, their computational time and memory requirements are important challenges for many applications. The idea of using the monocular image to capture depth information could potentially solve the memory requirement issue, but it is computationally difficult to capture the global properties of a scene such as texture variation or defocus information.

There has been a significant improvement in learning-based monocular depth estimation methods over the past couple of years. The majority of the deep learning-based methods involve a CNN trained on RGB-images and the corresponding depth maps. These methods can be categorized into supervised, semi-supervised and self-supervised. Supervised methods accept a single image and the corresponding depth information for training. In such a case, the trained network can directly output the depth information.

Let $I \in R^{w \times h}$ be an image with size w × h. The goal is to estimate the corresponding depth information $D \in R^{w \times h}$. This is an ill-posed problem as there is an ambiguity in the scale of the depth. Supervised learning-based methods try to address this issue by approximately learning the scale from a set of training images. On the other hand, unsupervised and semi-supervised methods often utilise an extra input for training such as stereo image sets, visual odometry and 6D camera pose estimation to tackle the scale ambiguity issue. These methods mathematically define the problem as follows: given a large dataset of Red-Green-Blue (RGB) and depth images, single image depth estimation can be considered as a regression problem that uses a standard loss function such as Mean Square Error (MSE). To achieve this, a training set τ can be represented as follows:

$$\tau = \{ (I_n, D_n) \} , I_n \in R^{w \times h} \text{ and } D_n \in R^{w \times h}$$

2.4 Related Work

A significant amount of computer vision based methods have been developed and implemented to solve motion tracking. Motion detection and tracking is a complex task due to dynamic parameters such as object luminance, speed, occlusion, scale, other stationary objects, camera motion etc. K. Kale *et al.* [5] proposed an object detection and tracking algorithm which is a combination of two computer vision techniques: optical flow and motion vectors estimation. Optical flow was used for object detection, it calculates the motion between two frames which are taken at different time intervals for every pixel in the frame. It is a discriminative method that poses objects to be tracked as a binary classification problem in a local image region, providing vectors that contain information about the object's direction and movement. The motion vector estimation technique provides an estimation of an object's position from previous frames which increases the accuracy of this algorithm and helps produce great results irrespective of image blur and cluttered background.

D. Zhang *et al.* [6] proposes a neural network tracker that combines convolutional and recurrent networks with RL algorithms. It formulates the visual tracking problem as a sequential decision-making process and proposes a novel framework, referred to as Deep RL Tracker (DRLT), which processes video frames as a whole and directly outputs location predictions of the target in each frame.This model integrates convolutional networks with recurrent networks , and builds up a spatial-temporal representation of the video. It fuses past recurrent states with current visual features to make predictions of the target object's location over time. The RL algorithm trains the model to maximize tracking performance in the long run. The neural network components are trained with backpropagation and the reinforce algorithm is used to train the policy network. The traditional CNN's are augmented with a recurrent convolutional model learning spatial-temporal representations and RL to maximize long-term tracking performance.

E. Ilg *et al.* [3] proposes a consolidation of the Flownet idea, FlowNet2 inherits the advantages of the original FlowNet, such as mastering large displacements, correct estimation of very fine details in the optical flow field, the potential to learn priors for specific scenarios, and fast runtimes. At the same time, it resolves problems with small displacements and noisy artifacts in estimated flow fields. This leads to a dramatic performance improvement on real-world applications such as action recognition and motion segmentation. Flownet2 is only marginally slower than the original FlowNet but decreases the estimation error by more than 50%. It

performs on par with state-of-the-art methods, while running at interactive frame rates. Another improvement introduced in the paper shows how stacking multiple networks using warping operation can improve results significantly. Varying stack depth and size of components results in many network variants with different sizes and runtime, this provides control over the tradeoff between accuracy and computational resources.

Huang [16] proposes a stereo reconstruction using CNN algorithms have been recently proposed . Prior work considered the subproblem that looks at image pairs , or three consecutive frames . Joint key-frame based dense camera tracking and depth map estimation was presented by. In this work, we seek to push the performance for single image depth estimation. We suspect that the features extracted by monocular depth estimators could also help derive better multi-view stereo reconstruction methods. stereo reconstruction using CNN algorithms have been recently proposed. Prior work considered the subproblem that looks at image pairs, or three consecutive frames. Joint key-frame based dense camera tracking and depth map estimation was presented by. In this work, we seek to push the performance for single image depth estimation. We suspect that the features extracted by monocular depth estimators could also help derive better multi-view stereo reconstruction methods.

Maaten [17] states that Transfer Learning approaches have been shown to be very helpful in many different contexts. In recent work, Zamir et al. investigated the efficiency of transfer learning between different tasks, many of which are related to 3D reconstruction. Our method is heavily based on the idea of transfer learning where we make use of image encoders originally designed for the problem of image classification. We found that using such encoders that do not aggressively downsample the spatial resolution of the input tend to produce sharper depth estimations especially with the presence of skip connections.

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.

We 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*. We will split this dataset into dedicated training and validation sets that follow a 85/15 percent train/ validation split. In other words, we 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 our 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 we 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.

3.2 Method and Implementation

Our approach employs multiple processes to estimate a vehicle's speed from monocular video input. In order to achieve more accurate speed estimations, we combine depth information with optical flow. For each sequential image pair we will first generate an optical flow value and a depth value. Next, we scale the magnitude of optical flow (OF) vectors, of a given image pair, by dividing by the mean of the disparity (depth) values in order to appropriately adjust the magnitude of the optical flow vectors calculated in the prior step. Finally, we compute the scaled speed estimates over the given sequence of frames.

A standard loss function for depth regression problems considers the difference between the ground truth depth map y and the prediction of the depth regression network $\hat{y}$. Different considerations regarding the loss function can have a significant effect on the training speed and the overall depth estimation performance. In our method, we seek to define a loss function that balances between reconstructing depth images by minimizing the difference of the depth values while also penalizing distortions of high frequency details in the image domain of the depth map. These details typically correspond to the boundaries of objects in the scene.

We start with a pretrained model on the KITTI dataset which contains a montage of images with their estimated depth maps. Our method expects dense input depth maps, therefore, we needed to run a depth inpainting method on the Lidar data. For our experiments, we use a Python re-implementation of the Matlab code provided by NYU Depth V2 toolbox. The entire 80K images takes 2 hours on an 80 nodes cluster for inpainting. For training we use the KITTI dataset with 4Gb GPU and batch size of 8. Testing is done on a subset of KITTI.

We implement our proposed depth estimation network using Torch and trained on four NVIDIA GTX 1660 ti GPUs with 6GB memory. Our encoder is a DenseNet169 pre-trained on ImageNet. The weights for the decoder are randomly initialized. In all experiments, we used the ADAM optimizer with learning rate 0.0001 and parameter values $\beta 1 = 0.9$, $\beta 2 = 0.999$. The batch size is set to 8. The total number of trainable parameters for the entire network is approximately 42.6M parameters. Training for the KITTI dataset is performed for 300K iterations, needing 9 hours to train.

We constructed our optical flow models 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, we 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 our dedicated training and validation splits constructed from *train.mp4* and *train.txt*, we set the hyper parameters to the following:

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

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

We 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, we 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 we stop training.

For those interested in viewing the full implementation of all models and our training and development environment, please feel free to access our code shared in this GitHub repository: [link][18].

3.3 Evaluation Strategy

Our goal is to generate real-time speed estimates with low error with respect to the ground truth values provided by comma.ai. We will use Mean Squared Error (MSE), the average of squared errors

We aim to both understand the baseline performance of each model (FlowNetSimple, FlowNetCorr and FlowNet2-CSS) with respect to the accuracy of speed estimations using dashcam data from Comma Ai and attempt to prove that a scaled speed that takes into consideration single-view depth information will produce superior results when compared to any one of the models individually. So, we would like to see lower MSE values for experiments that take into consideration monocular depth information.

For each unique experiment, we 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 our newly trained model. We will initially evaluate the performance of speed estimation methods using (FlowNetSimple, FlowNetCorr and FlowNet2-CSS), and then estimate MSE of scaled speed estimates that take into consideration flow estimates from (FlowNetSimple, FlowNetCorr and FlowNet2-CSS) and disparities from monocular depth information. We will get a non-negative MSE value, and our aim is to minimize it.

4 Results

4.1 Monocular Depth Estimation

We implemented the Depth model as detailed in section 3.2 and the depth estimation and the result was as shown. Since our loss function is designed to not only consider point-wise differences but also optimize for edges and appearance preservation by looking at regions around each point, the learning process does not converge well for very sparse depth images. while quantitatively our method might not be the best, the quality of the produced depth maps is comparable if not better than those produced by the state-of-the-art.

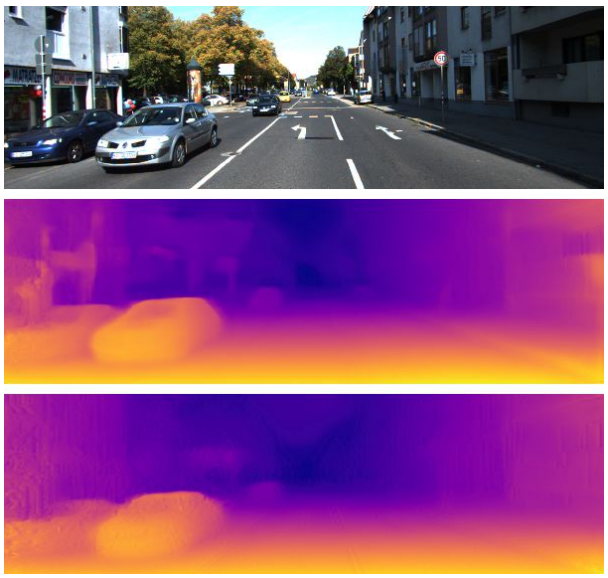

Figure 7: Top-input RGB image from the KITTI dataset, Middle-estimated depth map, Bottom- results of the model.

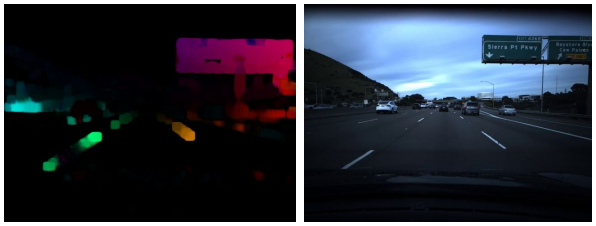

Figure 8: Detecting optic flow from sequential image inputs

4.2 Nvidia (Baseline)

We implemented the Nvidia CNN architecture [15] to generate baseline results in order to offer a performance comparison point for all subsequent FlowNet models. Note from figure 7 that after cycling through 10 epochs we were able to define a set of weights that achieved a minimum validation MSE of 3.46.

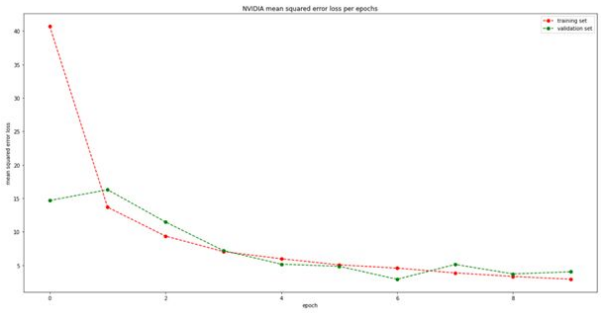

Figure 9: Mean squared error loss pre epoch - Nvidia model Validation MSE: 3.46

Training loss for the baseline Nvidia experiences a large decline of approximately 28 points in mse from the first epoc, at an mse of approximately 42, to the second epoch, at an mse of approximately 14. Thereafter, training loss steadily declines to a minimum mse at the tenth epoch. Validation loss starts at 15 at the first epoch and generally 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 at the sixth epoch and unfortunately does not improve thereafter. We take the updated weights of the eighth epoch and stop training early.

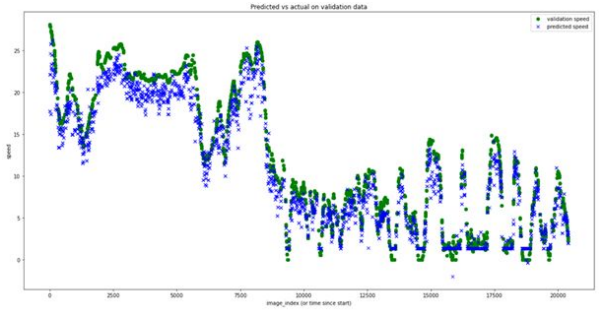

Figure 10: Observed (green) v.s. Predicted validation speed (blue)

From figure 8 above, we can see that the baseline model generally does a great job at predicting the ground truth speed of our dedicated validation set. We 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.

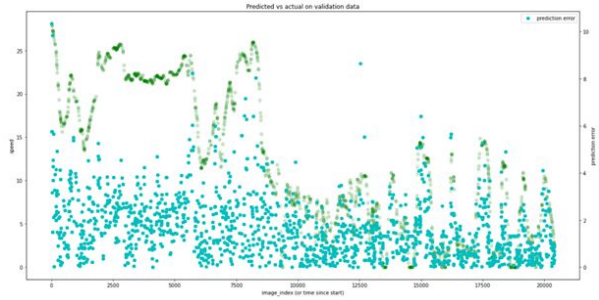

Figure 11: Predicted validation speed error (teal)

The minimum validation mse of 3.46 equates to an average speed estimation error of 1.86 miles-per-hour (mph). Seemingly impressive, but we can do much better!

4.2.1 Nvidia (Baseline) + Depth [Incomplete]

4.3 FlowNetS

Note from figure 10 below that after cycling through 13 epochs we were able to define a set of weights that achieved a minimum validation MSE of 1.25. In comparison to the baseline Nvidia

baseline model, FlowNetS was able to reduce the validation MSE by about 64%

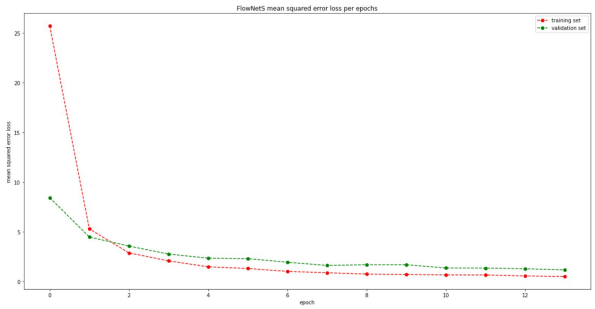

Figure 12: Mean squared error loss pre epoch - FlowNetS model

Training loss for the FlowNetS model experiences a large decline of approximately 21 points in mse from the first epoc, at an mse of approximately 26, to the second epoch, at an mse of approximately 5. Thereafter, training loss steadily declines to a minimum mse at the fourteenth epoch. Validation loss starts at about 8 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 at the thirteenth epoch and unfortunately does not improve thereafter. We take the updated weights of the thirteenth epoch and stop training early because at this point the training loss is at an mse of less than our minimum requirement to proceed to the next epoch.

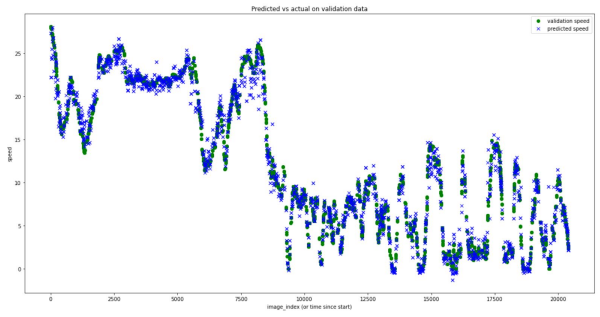

Figure 13: Observed (green) v.s. Predicted validation speed (blue)

From figure 11 above, we can see that the FlowNetS model does a great job at predicting the ground truth speed of our dedicated validation set. We 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.

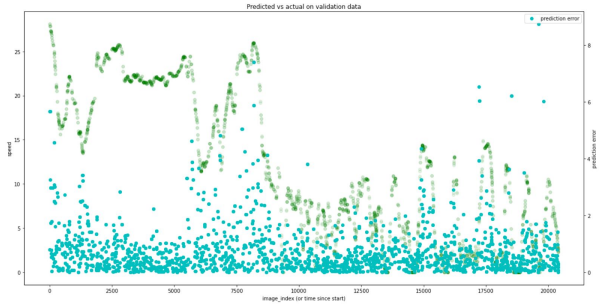

Figure 14: Predicted validation speed error (teal)

The minimum validation mse of 1.25 equates to an average speed estimation error of about 1.18 miles-per-hour (mph)!

4.3.1 FlowNetS + Depth [Incomplete]

4.4 FlowNetCorr

[Incomplete]

4.4.1 FlowNetCorr + Depth [Incomplete]

5 Conclusion

In this paper we initially set out to construct a system of state of the art optical flow and depth models for estimating a vehicle's speed via monocular dashcam video as input. We constructed our hypothesis based on the intuition that optical flow vectors can accurately model the speed at which the imaging device (monocular vehicle dash cam), and therefore the vehicle, is moving through space and that features within images that are closer to the vehicle seem to move faster than those features in the horizon. We first reconstructed the baseline Nvidia model to reproduce speed estimation results previously accomplished by members of the SpeedChallenge community.

With the baseline Nvidia model, we were able to produce better results at an mse of 3.46 after heavy edits to the data preprocessing methodology we used as a guide in constructing our development environment for all other flownet and flownet + depth implementations. Next, we constructed FlowNetS according to the architecture shown in figure 3 without the refinement network at the end. We 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 we were able to outperform the performance of the baseline Nvidia model at an mse of 1.25.

Unfortunately, due to time constraints and the loss of a teammate, we were not able to complete all that we initially set out to do. To be exact, first, though we were able to successfully reconstruct the FlowNetCorr architecture according to figure 4, again without the refinement network, however we were not able to train it properly without experiencing exploding loss values, a known phenomenon noted in [4]. Second, we took the professor's advice on reducing the complexity of our project and decided to not work on implementing FlowNet2(CSS). Third, we initially spent a considerable amount of time researching various ways to implement the depth model. We started with an implementation which did not give us the results we were hoping for, the depth model produced reconstructions that were too noisy. Which led to us switching gears to find a more advanced method with better performance as mentioned in section 3.2 and 4.1.

Ultimately, due to the unexpected loss of a teammate and the underestimated complexity of all that we initially set out to do in our project proposal, we were unable to successfully train our implementation of FlowNetCorr and did not get the opportunity to retrain all optical flow models in combination the with monocular depth information, thus we were not able to complete about half of our results section. However, despite our shortcomings, we stand by our initial hypotheses and strongly believe we will be able to produce more accurate speed estimates by scaling optical flow vectors with monocular depth information if given more time. Because of this, we intend on continuing to work on this project

throughout the next couple months to completely implement all FlowNet architectures and produce speed estimates using our depth model.

6 Individual Tasks

Tigran formulated the scope of the paper, designed, implemented, trained, and tested the baseline Nvidia and all modified FlowNet CNN architectures. Kuhoo researched various implementation techniques for monocular depth model, implemented, trained and tested said model. Both group members documented their respective contributions throughout this paper.

Kuhoo : Individual Tasks.
Initially me and our third partner resear

7 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] C. Godard, O. M. Aodha, and G. J. Brostow. Unsupervised monocular depth estimation with left-right consistency. In IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 6602–6611, 2017.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

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

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

[12] 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.
[13] LeCun, Yann; Bengio, Yoshua; Hinton, Geoffrey (2015). "Deep learning". Nature. 521 (7553): 436–444.
[14] 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.
[15] https://developer.nvidia.com/blog/deep-learning-self-driving-cars/
[16] 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.
[17] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pages 2261–2269, 2017.
[18] https://github.com/TigranMelkonian/Real_Time_Vehicle_Speed_Estimation