UNIVERSITY OF AMSTERDAM

MSc Artificial Intelligence

Track: Computer Vision

Master Thesis

Visual Speedometer: Learning Velocity from Two Images

by

C.K. Loor

10624635

May 3, 2017

42 ECTS
April 2016 - May 2017

Supervisor:
Dr. J.C. van Gemert
MSc. P.S.M. Mettes

Assessor:
Prof. Dr. T. Gevers

University of Amsterdam

Abstract

Predicting speed is something we do everyday; we walk, we drive, we pick up a pen. We are constantly keeping track of our movements. There has been much research into predicting speed. This has resulted into approaches that track wheel rotations and the development of different sensors, such as accelerometers. The research has shown reliable in the past and has been integrated into practice, whereas other research is new and has led to the development of new sensors and algorithms to predict speed. While there has been much research into predicting speed, no has been conducted to learn speed. This thesis focuses on learning speed from images, as cameras are cheap and the footage can be used for various purposes. Research in detecting movement between images quickly leads to optical flow and tracking of objects. In this research we will exploit a model that has learned optical flow to create our own model that has learned what velocity looks like. We prove that our model is capable of predicting speed accurately within areas that contain enough context. We investigate the limitations of our model to gain insight into our model's behaviour. Ultimately, our experiments prove that the amount and type of context in the images is integral to making a proper speed prediction from images.

Acknowledgements

I remember the first lesson of Machine Learning 1. I realised how hard I would have to work to achieve this title I had set my mind on. A few weeks later, I wondered why on Earth I decided to push myself academically for another three years. While studying I rediscovered the answer I already knew; science is wicked awesome. This has not been an easy ride. I have had to work hard, but I enjoyed most courses. And now here we are, at the end.

I would like to thank my supervisors Jan van Gemert and Pascal Mettes of the University of Amsterdam their feedback throughout this process. I could reach out to both of them whenever necessary and their responses were quick and clear.

I also want to thank the Aiir team. We have started an adventure together, during which graduating has had been pushed to the back burner. Nevertheless, their continued motivation and feedback on this research has been a great help.

Many thanks to Sharon, Elise and Iva for their support over the past three years and proofreading this thesis.

Thank you to my loved ones who I have slightly neglected during this research. Somehow, each time I tended to get too wrapped up in my work they were there to get me out of my head and relax before continuing my work.

Finally, I would like to thank you for reading this thesis. I have spent quite some time on this research and I have learned a lot along the way.

-Cassandra Loor, Amsterdam Wednesday 18 April 2017

Contents

Figure 1: Two images taken 0.1 seconds apart. The images originate from a stationary camera that is secured onto a moving vehicle. Can you tell how fast the vehicle is moving?
Source: [7]

1 Introduction

Imagine you're moving. You can be walking, running, biking, driving, as long as you are moving. We all know what it feels like to accelerate, to decelerate. We use this information to determine when and how much we need to slow down when approaching a traffic light. There are so many sensors in the human body that tell us we are moving; our eyes, our ears, the moving of fluids in our bodies. In other words, all indicators in our body contribute to an internal model we have that interprets how fast our body is moving.

What if we could only rely on our eyes to tell us we are moving? Would we still be able to tell that we are moving? Could we tell exactly how fast we are moving? In Figure 1, a camera was mounted to a moving vehicle. Can we, as humans, look at the two images and know how fast the vehicle is moving? Logically, to calculate velocity we need to know the distance travelled and the time between the two images. There is a sign on the left of the image. How far did it move? Did it move towards the vehicle or away from it? We can ask these questions about each object in the image and the changes in their location between the two images. We also need to take into account the camera itself. At what interval does the camera record an image? Under what angle is the camera recording images? How far off the ground is the camera? We can attempt to answer these, and many more, questions to try to determine the velocity of the vehicle. Chances are that our estimation is wrong. Though our eyes are excellent in detecting the differences between the images, we cannot accurately estimate how fast the vehicle is moving.

During this research we will investigate whether a computer can create its own representation of velocity from two consecutive images. We will learn velocity by exploiting an algorithm that was created to track movement, called optical flow. Within the field of Computer Vision, optical flow is used to detect and track motion in images. Many algorithms have been developed over time to detect optical flow and they work well for small

displacements. There has been much research to make optical flow algorithms robust to larger displacements, most of which are based on the same technical assumptions to track changes between consecutive images. Recently, a Convolutional Neural Network was successfully trained to predict optical flow. This model is known as FlowNet. Unlike other models, FlowNet does not require the traditional optical flow calculations or technical assumptions.

We will create a new Neural Network that exploits the knowledge of a network, FlowNet, that has learned displacements. Since velocity is defined as the displacement over time, we can exploit FlowNet to learn velocity. Since FlowNet has already explicitly learned displacements, our new Neural Network will only have to learn explicitly what velocity is from the displacements. We hypothesize this prior knowledge is an advantage and will lead to successful predictions of velocity.

Our main contribution is that we directly learn velocity from images. No extra calculations have to be performed to determine the velocity of the vehicle. Additionally, we will learn velocity from the vehicle's point of view (ego-vehicle speed). Traditional optical flow approaches are static cameras that track movement in the image. The same approach is used to track vehicles on the road. We place a camera on a vehicle and use those images to learn velocity. This will cause each pixel in the image to move, leaving our model to learn velocity when each pixel in the image has moved. Though much research was conducted into measuring velocity, this has led to new sensors such as accelerometers that can only measure velocity. With our approach, a camera can be used to determine the velocity of the vehicle. The images can be used to determine the velocity of the vehicle, but image analysis techniques can be used to determine the environment of the vehicle and more. To our knowledge, there has been no research into learning velocity from two consecutive images before.

If our model can learn what velocity looks like, it should also be able to learn what time looks like. Therefore, we also experiment with a secondary data set to evaluate how well our new Neural Network can evaluate images and map displacements to a time frame.

This thesis first discusses the core topics necessary to understand the concepts this research is based on in Section 2. Other research in the field of measuring velocity is discussed in Section 3. In Section 4 we will discuss our model and our design choices. The experiments and results are discussed in Section 5, followed by a discussion, conclusion and future work in Sections 6, 7 and 8, respectively.

Figure 2: An example of optical flow. The first two images are of a moving robot. The last image displays the optical flow; the direction and magnitude the robot moved into.

2 Background

The goal of this research is to learn velocity in two consecutive images. We hypothesize we can use optical flow to learn this. We want to use an existing model that has learned optical flow and optimize it to learn velocity. By applying Transfer Learning techniques, we will create a new Convolutional Neural Network. To do so, we will need extensive information on three core topics: Optical Flow, Convolutional Neural Networks and Transfer Learning.

2.1 Optical Flow

Optical flow is a term that was introduced by American psychologist James J. Gibson in the 1940s [8]. He used the term to describe the pattern of apparent motion. More specifically, the optical flow is described as *'the change of structured light in the image, e.g. on the retina or the cameras sensor, due to a relative motion between the eyeball or camera and the scene'*. An example is shown in Figure 2.

In the field of Computer Vision, optical flow is referred to when estimating movement, i.e. tracking, of objects in consecutive images, i.e. frames. Many papers have been written on optical flow and tracking algorithms. One of the most famous techniques to estimate movement is the Kanade-Lucas Tracker (KLT) [34], due to its simplicity and effectiveness. To determine the direction an object has moved into, the following assumptions are made: *the brightness constancy assumption* and the *the spatial smoothness assumption*. The brightness constancy assumption states that the intensity of a small region of pixels remains the same, despite a position change. The spatial smoothness assumption takes into account that neighbouring pixels tend to belong to the same object, and therefore move in the same direction. The KLT exploits changes in structured light; by keeping track of the intensity of a neighbourhood of pixels, a movement vector is associated to the pixels which contains the direction and distance the pixels travelled. This is displayed in the right image of Figure 2. The pixels which intensity

have not changed are the blue dots. The lines in the image indicate the direction the robot has moved into.

This means that, in essence, the KLT detects changes of pixel intensity between two images. Using the changes in intensity, the algorithm determines in which direction which pixels have moved. This is defined below in Equation 1. The intensity I of the pixel at location (x, y) moves in the x and y direction over time t. How much the pixel has moved into the x direction is determined by u, whereas the direction the pixel moved in the y direction is determined by v. If we solve Equation 1, we have determined the direction pixel (x, y) has moved in over time.

$$I_x(x, y) \cdot u + I_y(x, y) \cdot v = -I_t(x, y) \tag{1}$$

Though we now have a tool to track pixels over time using their intensity, the change in the intensity of a single pixel does not contain enough information to determine if there was a displacement or to give any indication about the displacement direction. Therefore, a neighbourhood of pixels is often used to determine the displacement of an object. We can do this under the spatial smoothness assumption. This usually leads to averaging the optical flow over a region of 3×3 pixels. The neighbourhood of pixels gives us an overdetermined system of equations with only 2 unknowns to solve, as shown in Equation 2. To solve this system, the Least Squares method is often used to determine an averaged optical flow across the region.

$$\begin{bmatrix} I_{x_1}(x, y) & I_{y_1}(x, y) \\ I_{x_2}(x, y) & I_{y_2}(x, y) \\ \vdots & \vdots \\ I_{x_n}(x, y) & I_{y_n}(x, y) \end{bmatrix} \begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -I_{t_1}(x, y) \\ -I_{t_2}(x, y) \\ \vdots \\ -I_{t_n}(x, y) \end{bmatrix} \tag{2}$$

2.1.1 Recent developments

The KLT, though it performs well, works only under the brightness constancy assumption and the spatial smoothness assumption. The KLT and other approaches to solve optical flow only work well for small displacements. To determine optical flow better, there has been research into different approaches that can predict optical flow over large displacements [35, 21]. Recently, there has been more research into large displacement optical flow. This research often goes hand in hand with making optical flow more robust.

Liu et al. [19] focused on scene alignment. To achieve this they decided not to rely on raw pixels, but on matching descriptors as these contain different information. A SIFT descriptor is densely sampled at a pixel level to extract local image structures and contextual information. To align the images of scenes, they match the SIFT descriptors which they refer to as SIFT flow. Their research has shown that matching of SIFT descriptors allows for alignment of images where objects are viewed from different viewpoints. The authors argue that their approach could be complementary to optical flow, but that SIFT flow cannot replace optical flow.

Work by Brox et al. [2] has combined the advantages of traditional optical flow and descriptor matching to allow for large displacement optical flow. The authors define regions in images. They extract features from each region and match the descriptors. This leads to a sparse set of correspondences. The sparse correspondences are integrated into a variational approach that leads to large displacement solutions. The authors prove that optical flow can benefit from sparse point correspondences, allowing optical flow to be applied to tasks where large displacements occur.

Weinzaepfel et al. [36] combine deep matching with a variational approach for optical flow. Like Brox et al. [2], the authors make use of descriptor matching. This is extended into an approach called EpicFlow [25]. Though these approaches use sparse convolutions and max-pooling to aggregate information from the features, these approaches do not learn optical flow and are not intelligent.

Machine learning techniques have been used to learn optical flow. Rosenbaum et al. [26] found that a Gaussian mixture model provides a better model for local optical flow statistics. Their model jointly learns the local intensity pattern and the local optical flow. Eventually, the model can distinguish between flat patches and boundary patches.

Sun et al. [30] formulate optical flow as a probabilistic inference problem. Their formulation allows them to learn the spatial smoothness and the brightness constancy. They extend their formulation by modelling the derivatives of optical flow. They trained on a small data set, since little ground truth information is available. The model performs well, but it is evaluated on a very small set making it difficult to determine how well this approach learns optical flow.

One of the most recent developments in learning optical flow comes from Dosovitsky et al. [6], where they formalize optical as a complete learning problem. They designed a neural network, called FlowNet, that learned optical flow from the image inputs. The models achieve competitive accuracy, proving that the authors are successful in learning optical flow.

Contrary to the papers we have discussed, optical flow is crucial in this research but not the goal. We will exploit FlowNet's knowledge of optical flow to create our own network. Our model will exploit the detected displacements to learn velocity, changing the traditional optical flow definition to create one of velocity.

2.2 Convolutional Neural Networks

A traditional Neural Network (NN) consists of many interconnected nodes. An example is shown in Figure 3. All nodes are organised into layers. Each extra node makes the model more complex, due to its interconnectedness. The input layer of an NN contains as many nodes as there are data points. In the case of images, each pixel is a data point. For a 300×300 image with RGB colours, the first layer would contain $300 \times 300 \times 3 = 270.000$ input nodes. To train a traditional NN to analyse the image, layers would have to be added to the network. Adding layers will lead to many more parameters, making the network difficult to tune and prone to overfitting.

To prevent this from happening, the Convolutional Neural Network (CNN) was designed. As CNNs are designed to analyse images, the architecture is designed to avoid

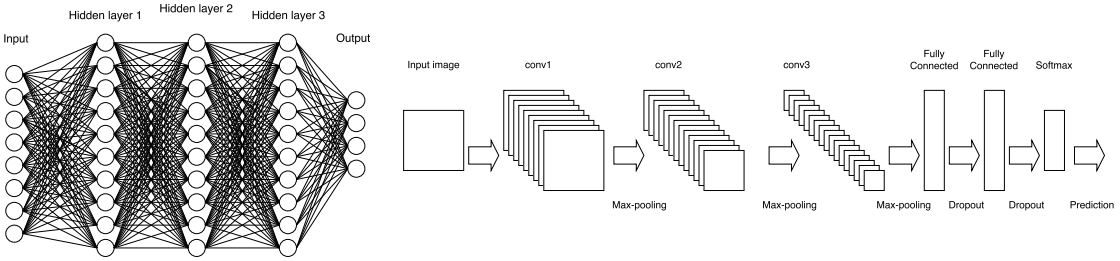


Figure 3: An example of what an NN and a CNN look like. On the right, an NN that consists of 8 input nodes, 3 hidden layers and an output layer. On the right, a simple CNN that consists of 3 convolutional layers and a Softmax function to determine the output.

the interconnectedness of a classical NN. In a CNN, an image is fed to the input layer. Instead of a node connected to each input pixel, there is a node for a subregion of the image. Each node in the convolutional layer is a filter. Applying this filter to an image, called a convolution, results in a new filtered image. The filtered image is then passed to the next layer for further analysis. This is the essence of CNNs; to convolve the image and create new, filtered images. To make CNNs more effective, other actions are also performed on the images, which allows CNNs to analyse images. The actions and layers unique to CNNs are discussed in this Section.

2.2.1 Layers

The most powerful layer in a CNN is the *convolutional* layer, which filters the image. Then there are the *ReLU* and *pooling* layers that make learning more efficient and detailed. We will discuss these layers in depth. Then, we will give an example of how CNNs can be built.

Convolutional layer The convolutional layer performs convolutions on the images. Convolutions are filters that are applied to the image. These filters can be used to detect vertical lines, horizontal lines, corners and more. The number of filters in a convolutional layer is determined by the architect of the network. When an image passes through a convolutional layer, each filter is applied to the image. This leads to the same number of convolved images (features) as there are filters in a convolutional layer. An example can be found in Figure 4. On the right, we see the input image that is being convolved with the filters in the first convolutional layer. The second image to the right displays the extracted features from the input image after the first convolutional layer.

Rectified Linear Unit CNNs, like traditional NNs, use backpropagation to train the network. Compared to the sigmoid activation function, the Rectified Linear Unit (ReLU) is very efficient in computation, while still allowing the networks to generalise well. Since ReLU does not allow negative values, chances of a vanishing gradient is

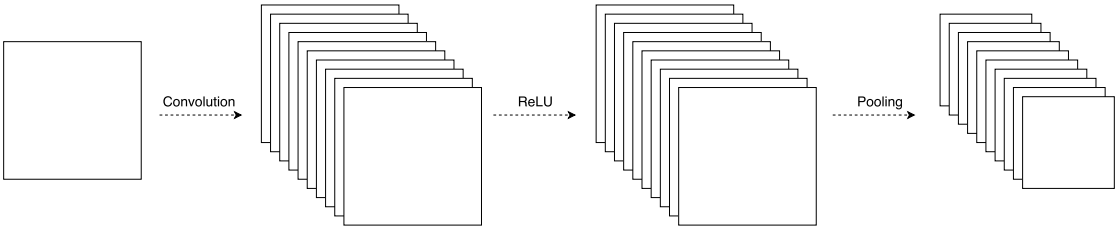


Figure 4: An example of the effects of the layers in a CNN. The first image is the input image, which is convolved with all features in the first layer. Then, the ReLU adjusts the features by changing all negative values to zero. After ReLU, pooling is performed which makes the the convolved images smaller.

reduced significantly. Another reason ReLU is used over an efficient linear function is because networks need non-linearity to compute non-linear functions, as not all tasks have linear solutions. This is a problem, since not all tasks have a linear solution. The use of the ReLU activation function is not specific to CNNs. It is also used in other NNs due to its effectiveness and quick training time. As shown in the middle of Figure 4, applying ReLU does not affect the size of the image.

Pooling To gain more information from the image, we want to perform more convolutions. However, this will be expensive if we continue to create convolved images per filter. A trick to create a more compact representation is to apply *pooling* [28]. Pooling means that we will take a sliding window over the convolved image and create a smaller image with the information from the sliding windows. Usually, this is a small window of 2×2 or 3×3, with a *stride* as large as the sliding window. Within our sliding window, we generally pick the highest value and place this in the new image. This leads to a much smaller image, which still contains essential information about the input image. The smaller image can now be used to extract more information from the image. Pooling also creates a model that is more robust to noise and helps achieve invariance to image transformations.

Picking the highest value in the sliding window is called *max pooling*. This technique is widely used. Theoretically, the architect of the CNN can also choose to pick the average value, *average pooling*. An example of max pooling using a 2-by-2 sliding window is shown in Figure 5. More about pooling can be found in the research by Boureau et

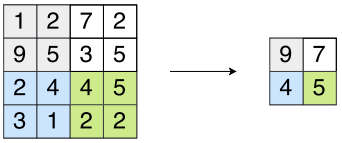


Figure 5: An example of max pooling. Using a 2-by-2 sliding window, the highest value in the window is used to create a new image. This is repeated over the entire image, downsizing the image while maintaining its features.

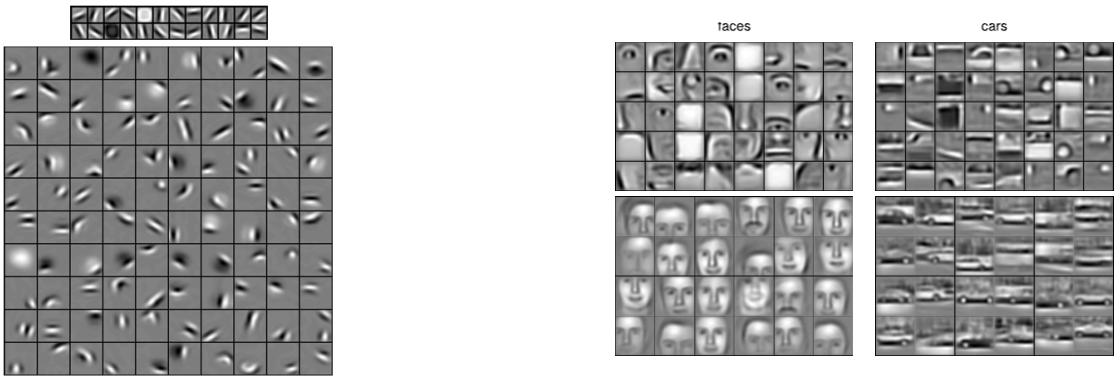


Figure 6: An example of what CNN layers learn. The image on the left displays what filters in a convolutional layer look like. The top filters are from the very first layer, where the filters at the bottom are from the second layer of the network. The image on the right displays the feature response of the second layer. The images at the bottom display the feature response of the third layer.
Source: [15]

al. [1].

As pooling downsizes the image, the features extracted in each convolutional layer entail a larger part of the input. As can be seen in Figure 6, the features of the first layers analyse part of the input image. The deeper the layer, the smaller the image, the more the features can *see*. This gradually leads to a CNN learning to see eyebrows, eyes, a nose and a mouth in the first layer and an entire face in the last layer.

Building a CNN Building a CNN is quite straightforward. The layers we introduced can be stacked to create a CNN. Depending on the size of the input image and the information needed, the number of layers (depth) of the CNN is determined. As the image becomes smaller after pooling, more information can be drawn from the image. As the image starts off large, the features drawn from there are different from the features of deeper layers. The first few layers tend to detect simple edges. The deeper layers tend to be able to detect more complex shapes, due to the application of filters on smaller images. An example is shown in Figure 6. The left side displays the filters that have been learned by their CNN. The filters are sensitive to edges, corners, and more. On the right side, the input images are displayed as *feature responses*. These feature responses is what a layer sees. The top images come from one of the first layers in the CNN. The bottom images come from some of the last layers of the CNN. In the first few layers, only parts of the image are detected. For the face, this means eyebrows or a nose and for the car it means the detection of a door or a wheel. Due to convolutions and pooling the image becomes smaller, but the essence of the image is maintained. Therefore, the last layers detect an entire face or car.

2.2.2 Recent developments

Since the design of neural networks, many different architectures and techniques were introduced. These architectures have led to neural networks that are specialised in different tasks.

An example of a different network architecture is the Recurrent Neural Network (RNN). RNNs contain an internal memory and use it to process sequences of input. The network then learns the connection between the inputs. RNNs have been used in speech and handwriting recognition to interpret letters and sounds and their meaning. Work by Sutskever et al. [31] has proven that RNNs can be used to learn how to translate sentences. Other work by Sak et al. [27] proves that RNNs are good at learning sequences of the sounds that make up speech. The results achieved by RNNs shows that they can learn the relation between sequences. If we regard an input of consecutive images, it is possible that they can also learn speed from images.

Within the field of image processing, the Fully Connected Neural Network (FCN) was introduced to predict dense outputs from input of arbitrary size. These networks only consist of convolutional layers. Long et al. [20] were the first to train an FCN end-to-end for pixelwise prediction and to train an FCN from supervised pre-training. The FCN is built from convolutional layers only, creating a deep filter. The built FCN uses a skip architecture that combines information from earlier layers with the output to create detailed segmentations on a pixel-level. The work by Long et al. has shown that FCNs can be used for image segmentation on different scales, due to its architecture.

There have been other uses for FCNs as well. DenseCap by Johnson et al. [12] is used to both localize and describe salient regions in images, in natural language. Dai et al. [5] have trained an instance-sensitive FCN to determine instances on a pixel-level. Another example is R-FCN, an FCN that performs object detection via region-based FCNs [17]. FCNs have shown to be powerful at making per-pixel predictions.

To conduct our research, we have chosen to fine-tune FlowNet. However, we could have also opted to train an RNN from scratch for its strengths to learn a relation of a sequence of inputs. We have chosen to mainly learn velocity from optical flow as displacements contain much information about velocity. Since FlowNet has knowledge of displacements, we have chosen to exploit this knowledge rather than train an RNN from scratch. In this research, we do not include results of RNNs. To our knowledge, RNNs have not been trained to learn velocity. Additionally, RNNs learn sequences whereas this research focuses on learning velocity from two images. We also include research into FCN's as FlowNet is based on the work of Long et al. [20]. The research has made it possible for the authors of FlowNet to learn optical flow and project the prediction on a pixel-level.

2.3 Transfer learning

Training neural networks from scratch is both difficult and time-consuming. Several millions of examples of data is required and the neural network needs to learn from the data for several epochs, which takes time. However, it often occurs that not enough

data is available for deep learning. In these cases we make use of transfer learning. We exploit the *knowledge* of existing models to still achieve our goal. To find out how transfer learning can take place, there are tasks and domains. The domains consist of a feature space and a marginal probability distribution. The task consists of a label space and an objective predictive function. This allows us to denote our transfer learning problems into that of a domain and a task. We generally wish to perform transfer learning when the source domain and the target domain differ or where the source task and the target task differ. Though it is also possible to use transfer learning when both the source and the task domain differ, but the architect would need a lot of data and the network should be able to learn its task. Therefore, transfer learning generally does not occur when both the source and task differ too much from the original application.

Pan and Yang [22] have conducted research into transfer learning. They have introduced an approach to transfer learning by asking what to transfer, how to transfer and when to transfer. This helps gain insight into what knowledge needs to be transferred. They also look into different transferring techniques. Their research is relevant as in this research we will perform transfer learning. Analysing our source domain and task domain, as well as the source task and the target task will help us understand which knowledge to transfer to our new network.

Chu et al. [4] have performed research into best practices when fine-tuning visual classifiers to new domains. They find that a large amount of data when fine-tuning always works best. If little data is available, the difference between the source model and the goal becomes more important. This is relevant to our research as we have limited data available. This research will help determine the learning rate of the individual layers we will copy to learn velocity.

Some of the most recent work has researched into reusing parts of a neural network. Work by Yosinski et al. [37] has shown that, in CNNs, the first layers often contain similar features across different trained models. This shows that the decisions in CNNs depends more heavily on the deeper layers in the models. They have shown that it is possible to freeze the first few layers of a trained network and fine-tune deeper layers, completely retrain deeper layers or even initialize a new network with the layers of a trained network and get better results. This is particularly relevant to this research as we will not simply fine-tune an existing network. We will have make changes to its architecture.

3 Related work

This section focuses on research that has been conducted to determine the velocity of a vehicle. There several ways velocity can be determined using a camera: by observing a vehicle from the side of the road or by tracking the velocity from within the vehicle. Additionally, there are many different sensors that can be used to determine speed. In this section we discuss research that is related to determining the velocity of a vehicle. We focus on research in roadside speed measurement and ego-vehicle speed measurement.

3.1 Roadside speed measurement

Litzenberger et al. [18] have created an embedded optical sensory system for speed estimation. They take an SR sensor, which responds to changes in illumination, and mount to traffic signs above the road. The SR sensor is vulnerable to changes in illumination, causing it to fire if a vehicle drives by. The signals are temporarily stored in Address Event (AE) data streams. Using digital signal processing techniques, the changes in illumination are detected. Based on the illumination clusters, the velocities are calculated and averaged to determine the speed of the vehicle.

Taghvaeeyan and Rajamani [32] developed a portable roadside magnetic sensor for vehicle counting, classification and speed measurement. To determine the speed of vehicles, they calculate the cross-correlation between longitudinally spaced sensors. They can calculate this quickly by using frequency-domain signal processing techniques. The advantage of their sensor is that it is portable and can be used along any road.

Jeng et al. [10] developed a side-looking single-beam microwave vehicle detector (VD) to estimate vehicle speed and length. The VD is placed at the side of the road with a squint angle. When a vehicle passes through the beam, the authors keep track of the single-beam signal as it is reflected back. From the signal, the authors extract sample range data. The authors apply Doppler FFT to the sample range data to obtain the Doppler frequency of the vehicle. This allows the system to determine the speed of the vehicle.

Lan et al. [14] have adjusted the traditional optical flow method to use optical flow and determine the speed of a vehicle. They use a camera, mounted above a road, to capture footage. The authors grab three frames. They derive difference images between f_{t-1} and f_t, and f_t and f_{t+1}. The two difference images are used to create a new image. They extract the vehicles from the image using local adaptive thresholding. After extracting the vehicles from the images, the authors calculate the optical flow. They suggest to use their newly developed gray constraint optical flow algorithm over the traditional optical flow algorithm, as their method does not assume that the gray value of a pixel is constant over time. After determining the optical flow of the vehicles, the speed of the vehicles is calculated. Using the resolution of the images and the width of the lane, the authors determine a displacement per pixel in meters. This allows the authors to express the displacement of the vehicle in pixels and calculate the speed, based on the image resolution and time between frames.

For this research the camera will be mounted to a vehicle, which will cause each pixel

in the image to change location. The use of a camera also removes the need to design a new sensor. Though we will use knowledge of optical flow, as Lan et al. did, we will use this knowledge to learn a representation of velocity.

3.2 Ego-vehicle speed measurement

Yu et al. [38] introduce SenSpeed, an application that uses the driving conditions to determine vehicle speed when the GPS signal is not reliable. They use an the accelerometer and gyroscope of a smartphone to achieve their goal. To determine the speed of the vehicle, the accelerometer is used. The authors came to find that the error of the accelerometer increases linearly over time, regardless of the smartphone model. To correct the accelerometer measurements, the authors find points in time where they are certain of the vehicle velocity. Using the gyroscope, the authors determine *reference points*. The reference points infer if the vehicle is turning, stopping or passing through bumps or potholes. Since the accelerometer error increases linearly, the authors can determine the error in accelerometer measurements accurately between two reference points.

Levefre et al. [16] have made assumptions about possible driving speed behaviours. They have formalised these models as parametric and non-parametric models. The models assume: constant speed, constant acceleration, keeping a safe distance, keeping a desired distance, a Gaussian Mixture Regression model and a Neural Network. The models were trained to optimize the parameters, based on the training set. The training set consists of a 77 GHz Delphi Electronically Scanning Radar which provides measurements of distance as well as relative speed to the vehicle in front. The actual speed and acceleration of the vehicle is extracted from the vehicle itself. The results have shown that simple parametric models work well for short-term speed prediction and that the more complex parametric models work well for long-term predictions. The non-parametric models work equally well or better than the parametric approaches.

Qimin et al. [24] use sparse optical flow to determine the speed of a vehicle. They mount a camera to a vehicle and drive it on flat terrain. The images that are captured by the camera are preprocessed and the contrast is increased to optimize the results of the algorithm. They use the KLT to determine the keypoint matches between the two images and use RANSAC to filter out incorrect the matches. The displacement of the vehicle is calculated by the difference in coordinates between the previous and current frame. The displacement of the vehicle in the world is calculated with pixel displacement and camera calibration parameters. By averaging the calculated speed over all matched keypoint pairs in the entire image, a vehicle speed is determined.

In this research, we will use two consecutive images to determine the velocity of a vehicle. Different from SenSpeed is that we will use a mounted camera, instead of a smartphone. Lefevre et al. have attempted to model driving behaviour, but this is based on radar information instead of images. Qimin et al. adhere a similar approach to ours; they mount a camera to a car and use the optical flow and camera calibration parameters to determine the velocity. Our approach does not require the knowledge of camera parameters or any extra calculation. Our approach only requires images to determine velocity.

4 Methodology

The goal of this research is to learn velocity from two consecutive images. We use two consecutive images from a camera attached to a vehicle and we approach this problem as a learning problem. Using FlowNet, a CNN that has learned optical flow, we transfer its knowledge to train our new CNNs to predict velocity. If we were to train our CNNs from scratch, the models would have to learn a relation between the displacements in the images and a velocity. By transferring knowledge of optical flow, we hypothesize information about displacements is readily available. This leaves our new CNNs to mainly learn a relation between displacements and velocity.

There are several approaches that measure the velocity of a vehicle using a roadside camera. Therefore, we perform a similar experiment where we will use a static camera to learn time. This problem still requires our models to map a displacement to a value, allowing us to design a network architecture that can be used to predict either time or velocity. This experiment will allow us to determine if the base model, FlowNet, can be exploited to learn velocity.

Though measuring time and velocity are similar problems, we will evaluate our models separately. We have formulated a classification problem to learn time and a regression problem to learn velocity. To learn time, we will use a stationary camera that observes an object moving across the image. To learn speed, we will mount a camera to a moving vehicle and learn velocity from the captured images. This leads to two different data sets: a set where an object moves in a scene, and a set where the camera moves through the scene. In the first data set only the object has to be tracked, whereas in the second each pixel will move. In the case of learning time it is imperative that each object moving across the screen moves with a similar velocity at a similar distance from the camera. If this is not the case, our model cannot learn time from images. When learning velocity, the restrictions are much less, though the frames passed to our model should always record at the same interval and remain mounted to the same side of the vehicle.

We will first discuss FlowNet and our own models in-depth. We will then discuss our design choices and introduce our models that will predict time and velocity.

4.1 FlowNet

This research is based on a neural network that has learned optical flow, called FlowNet [6]. Though there are many techniques that can calculate optical flow, FlowNet is the only known model that has learned it. FlowNet was trained on the Flying Chairs data set, which was generated by the authors, and fine-tuned on the MPI Sintel data set [3]. The authors have come up with two different network architectures: a Simple architecture and a Correlation architecture. Both architectures are based on the Fully Convolutional Network, meaning that this network can use any input image size to determine its optical flow. This is valuable as the calculation of optical flow can take place on images of arbitrary size.

Both FlowNet models contain a *contractive* part and a *refinement*. When performing convolutions on the images, the images are downsampled. This allows the network

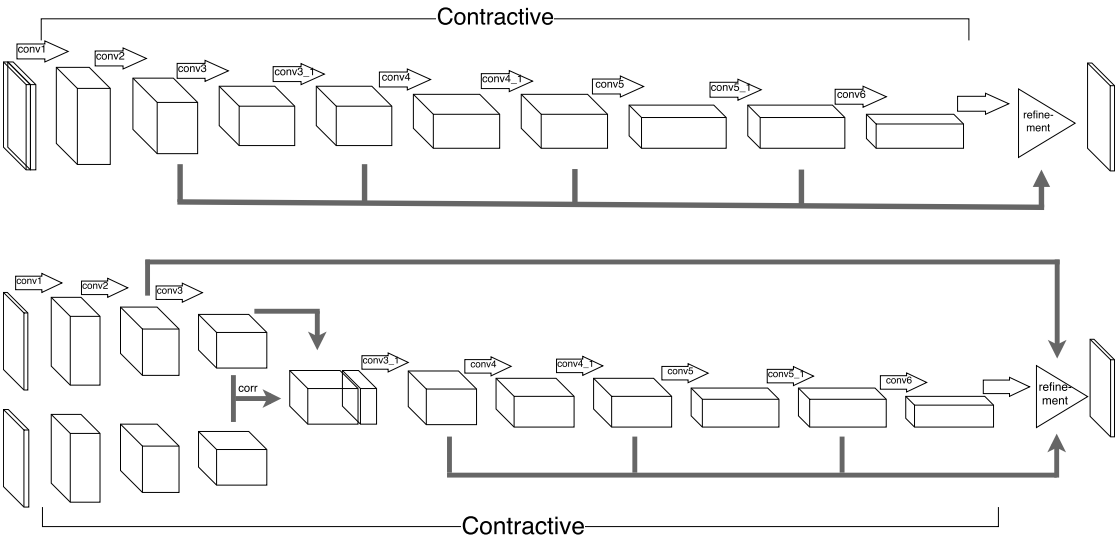


Figure 7: Schematics of FlowNet architectures. FlowNetSimple at the top and FlowNetCorr at the bottom. The refinement is the upconvolution of the image, which restores the image to its original size and returns the optical flow per pixel.

to learn high-level features, but also features that make sense of sub-regions in the image. The information from the downsized image has to be used to determine a dense per-pixel optical flow on the original image. This occurs in the contractive part of FlowNet. To retrieve optical flow per pixel, the authors implemented *upconvolutional* layers. These layers consist of both *unpooling* and a convolution. The upconvolutional layers are applied to feature maps and concatenate it to corresponding feature maps of the contractive part of the network and an upsampled coarser flow prediction. This allows for preservation of information from both high-level as well as lower-level feature maps. The authors decided to perform the upconvolution 4 times as more upconvolutions do not improve their results. The evaluation of the models have shown that, in general, FlowNetCorr outperforms FlowNetSimple.

4.1.1 FlowNetSimple

To train the Simple network, the two images are stacked together, leading to a 6-dimensional image. The following layers have been designed as a generic network, to let the network decide on how to learn optical flow. This architecture is shown in Figure 7 at the top. The authors hypothesize that if this architecture is deep enough, it could learn optical flow. However, it is difficult to determine beforehand if this network will ever succeed in truly learning optical flow since FlowNetSimple has a generic architecture. This makes the Simple model a shot in the dark and it may not be very reliable.

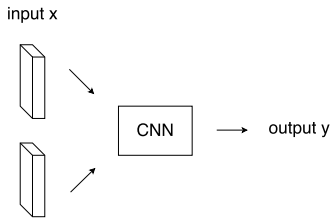


Figure 8: A schematic of our goal. We want to pass two images as input to a CNN and a prediction of speed or time, depending on the model we use.

4.1.2 FlowNetCorr

For the Correlation network, the authors decided to combine the two images at a later stage, after the model has created meaningful representations of each image separately. This lead to two separate, but identical image processing streams that perform a few convolutions on the images separately. To help the model learn the correspondences between the feature representation of the images, the authors introduced a *correlation* layer. This layer performs comparisons between each patch of the two feature maps. To explain the correlation, we consider a single comparison of two patches. The patches are centered at $\mathbf{x}_1$ in the first feature map, and $\mathbf{x}_2$ in the second map. For a square patch $K := 2k + 1$, where K is the size of the patch we will correlate pixel with each other, the correlation is then defined as follows:

$$c(\mathbf{x}_1, \mathbf{x}_2) = \sum_{\mathbf{o} \in [-k,k] \times [-k,k]} \langle \mathbf{f}_1(\mathbf{x}_1 + \mathbf{o}), \mathbf{f}_2(\mathbf{x}_2 + \mathbf{o}) \rangle \tag{3}$$

Equation 3 shows a convolution of data with other data. Therefore, this layer has no trainable weights and can be used for any image of arbitrary size. However, comparing each patch of each image with each other leads to $c \times K^2$ calculations. This is computationally expensive, making forward and backward passes intractable. Therefore, the authors have chosen to limit the maximum displacement.

4.2 Contributions

We aim for a solution as shown in Figure 8. By passing two input images x through a CNN, we get a prediction of velocity or time between frames. In the case of time this prediction will be one of the classes $k \in K$ and in the case of speed this prediction is a continuous real value. Since we investigate a time experiment (classification) and a velocity experiment (regression), we have to create two different models. The models are TimeNet, to predict the time between frames, and SpeedNet, to predict speed.

We hypothesize we can use FlowNet as a base model to predict velocity and time, due to its knowledge of optical flow. Optical flow determines the displacement of neighbourhoods of pixels. Since velocity is essentially the measurement of displacement over time, we hypothesize that FlowNet's knowledge already solves part of the problem for us. We want our new model to build upon FlowNet's knowledge of displacements to

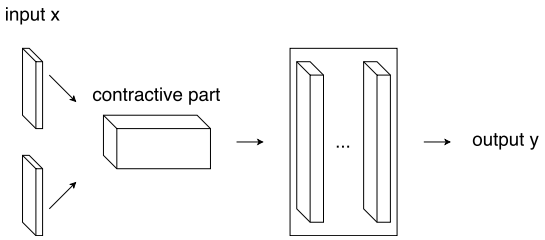


Figure 9: Schematic of the modified FlowNet architectures. The contractive part of the FlowNet models remain and the refinement was replaced with one or more fully-connected layers. The output y of the models depend on whether the models predict time or speed. If time is predicted, the model outputs a class. If speed is predicted, the model outputs a float value.

predict time and speed. This is a difficult task, as several objects in the images can move. This causes locally different optical flow, for which our model will have to derive a function that is able to determine which characteristics contribute to predicting speed or time.

As explained in Section 4.1, with each convolution the image is reduced in size. In the last convolutional layer of the contractive part in FlowNet, *conv6*, the optical flow of the image has been determined albeit on a reduced image size. From this point onward, the optical flow information from *conv6* is deconvolved to create the optical flow of the full-sized image. Since the refinement only deconvolves the down-sampled image and the optical flow to the full image, the refinement does not offer more information about the optical flow. Therefore, we do not need the refinement when learning velocity.

To create our new models, we will apply techniques of transfer learning. Our models only need the *contractive* part of the original FlowNet models. Therefore, we will copy the layers of the contractive part, along with their weights, into our new model. This leaves us with a model that can take input images x, convolves the images and outputs the features of the *conv*6 layer. For our new model to output our desired class $k \in K$ or real-value speed, we will add one or more fully-connected layers after the *conv6* layer.

The fully-connected layers contain 1024 parameters to increase the complexity of the loss function. We chose 1024 parameters, because the *conv6* layer outputs 1024 parameters. Additionally, simply outputting the results of the *conv6* layer to an output layer would result in a direct mapping of the activations of that layer. This is equal to linear regression and the problem at hand is too complex to solve with linear regression. By adding complexity, our models should be able to map the two images to a speed or time.

The architecture of the new models is depicted in Figure 9. The input images x are passed to the *contractive* part we have kept of FlowNet. The extracted features are then passed through the fully-connected layers, which yield the output y.

4.2.1 TimeNet model

In the case of predicting the time between frames, we have created a model called TimeNet. TimeNet is a classifier that can predict 1 of 6 different times between frames in ms (classes). To create the classifier, we use Caffe's [11] *Softmax with Loss* layer. This layer performs a generalisation of the binary logistic classifier to a multi-class classifier by using the Softmax function and outputs a probabilistic prediction of the classes.

By passing the input x through TimeNet, the last layer will use the features from the last fully-connected layer to assign a probability to each class. This is outputted as a vector, y, that has the length of the number of classes. In this case, this vector has a length of 6. The output of the Softmax classifier is normalized, ensuring that the output vector contains values between 0 and 1, and sums to 1. This is the predicted distribution over the classes. We formalize the prediction of TimeNet as $P(y|x)$.

$$P(y_k|x, W) = \frac{e^{f_{y_k}}}{\sum_{k=1}^{K} e^{f_k}} \tag{4}$$

Equation 4 formalizes the Softmax classifier. The classifier itself cannot be trained, it only translates the output of TimeNet to a probability distribution. For this reason, we take into account the weights matrix W when we formalise the Softmax classifier. This leads to Equation 4, where y_k for $k \in K$ is the predicted probability per class, f_k is the actual probability per class, x is the input images and W are the weights. The class with the highest probability mass is the prediction of TimeNet.

During training time, TimeNet learns to interpret the features of the *conv6* layer to make a good prediction. The model receives the input x with a target label t. The Softmax classifier itself cannot learn how to interpret the features, it can only convert the output of the model to a probability distribution. However, during training time we can adjust the weights in W which is why they are explicitly mentioned to the parameters in Equation 4. This allows us to adjust the weights of our model during training in such a way that the Softmax classifier's output is correct. We feed TimeNet the input image pair x and the label t. The target label t is a one-hot vector, as we only have 1 class that can be correct and thus the entire probability mass is in the correct class. By passing x through TimeNet, the prediction of y will be poor at the start of training. Using a *loss function*, we can determine how wrong the prediction y is from the target t. This information is then used to perform backpropagation, which adjusts the weights of TimeNet. In essence, during training time the model learns correct behaviour. After adjusting the weights, we evaluate the predictions of TimeNet again and we again adjust the weights accordingly, until the desired behaviour is achieved.

To determine how the weights of TimeNet must be adjusted during training, we use the cross-entropy loss function. The cross-entropy loss function is depicted in Equation 5, where L is the loss, the classes are $k \in K$ and y_i is the class prediction by TimeNet. The cross-entropy collects the mean of the losses per class. If the loss increases substantially,

the model performance decreases and thus more weight updates should be performed. Using backpropagation, the model will update its weights accordingly.

$$L = -\log\left(\frac{e^{f_{y_i}}}{\sum_k e^{f_k}}\right) \tag{5}$$

Using cross-entropy for a loss function has to do with the Kullback-Leibler divergence (KL-divergence). From an information theory point of view, the KL-divergence can be rewritten into terms of entropy ($\mathrm{H}(p)$) and cross-entropy ($\mathrm{H}(p, q)$), see 8. The KL-divergence is a measure to determine the difference between two distributions. In our case, there is a distribution within our data that we must learn, which we will refer to as distribution p. We assume a prior probability distribution q. We use the KL-divergence to measure the difference between distributions p and q. The difference between the distributions is used to update the weights in our network. After updating our prior probability distribution q, we again calculate the KL-divergence to determine how much information was gained and how much our weights need to be updated again to achieve our target distribution p. This process is continued until the prior probability distribution q and the target distribution p are the same. Essentially, the cross-entropy is attempting to create a probability distribution that is equivalent to the target distribution, trying to force TimeNet to assign all its probability mass into the correct class during training time.

$$\begin{aligned} D_{KL}(p||q) &= \sum_{k=1}^{K} p_k \log \frac{p_k}{q_k} && (6)\\ &= \sum_{k=1}^{K} p_k \log p_k - \sum_{k=1}^{K} p_k \log q_k && (7)\\ &= \mathrm{H}(p) + \mathrm{H}(p, q) && (8) \end{aligned}$$

4.2.2 SpeedNet model

To predict speed, we have created a model called SpeedNet. SpeedNet is a regressor that outputs a speed prediction in the form of a single floating number. To create the regressor, we again use Caffe. This model is different from TimeNet, as we no longer work with a fixed number of classes or output a probability distribution. If we were to continue working with classes, we would need a class to define each speed, which is difficult to achieve. Therefore, we want SpeedNet to output a single, floating value that is the predicted speed of the input image pair. To do so, we use regression. Using regression, SpeedNet can, theoretically, learn to predict speed even from image pairs it has not seen before.

To train SpeedNet, we no longer need to translate the output of our model to a probability distribution. During training time, we still feed an input image pair x to

SpeedNet, along with the target speed t. SpeedNet's prediction y is again evaluated by the loss function to adjust the weights of SpeedNet. To train the model properly, we again have to determine a loss function. For this experiment, we use Caffe's *Euclidean Loss* layer. The Euclidean Loss layer calculates the Sum-of-Squares error between the SpeedNet prediction y and target value t. The error between the prediction and the target value is then used to update the weights, using backpropagation. The Euclidean Loss function is formulated in Equation 9, where L is the loss, N is the number of samples, y is the value predicted by SpeedNet and t is the target value. The error is averaged and then used in backpropagation to update the weights.

$$L = \frac{1}{2N} \sum_{i=1}^{N} ||y - t||_2^2 \tag{9}$$

4.2.3 Training specifications

Aside from the architecture and loss functions, there are other training details that make up SpeedNet and TimeNet. We have attempted to keep these as similar as possible between the two models. Though there are some differences between the SpeedNet and TimeNet models, there are no training differences between the Correlation and Simple models of SpeedNet and TimeNet. We discuss these training details in this section.

Learning rate When training FlowNet, the learning rate was set very low. When fine-tuning the new models, the learning rate of the copied layers is set to $1e-4$ whereas the overall learning rate is set to 0.001. The learning rate in the fully-connected layers is set to 1. This allows the layers to slowly adjust themselves to the new data set during training. Every 5 epochs, the learning rate decays by half. In total, both SpeedNet and TimeNet are trained for 30 epochs.

Optimizer We train TimeNet and SpeedNet with the Adam optimizer [13]. The Adam optimizer [13] has shown quick convergence rates, training models quicker than other optimizers. It has been shown shown that there are two parameters that have to be set: β_1 and β_2. The paper suggests to initialize the values $\beta_1 = 0.9$ and $\beta_2 = 0.999$. These settings were used to train both SpeedNet and TimeNet. Though there are other optimizers, Adam has shown much success and effectiveness in training the FlowNet base model as well as numerous other models.

Batches The models were trained using mini-batches. Mini-batches contain a number of samples the model derives update gradients from. In a batch, the gradients are averaged and the averaged gradient is used to update the model. This leads to a smaller variance in the update gradients. If we were to update the model on each sample and the sample is not a proper representation of the whole data, the update will lead to noise in the model. Given this knowledge and the GPU constraints, we use a batch size of 50 image pairs for TimeNet and a batch size of 20 image pairs for SpeedNet.

Correlation layer parameters The Correlation model has its own parameters that can be set. These parameters affect the *Correlation* layer, which calculates the displacement of the pixels. It is possible to set the size of the displacement, the size of the patch and the strides. We have not adjusted these values for this research, maintaining the settings described in [6].

5 Experiments

In this section we evaluate our TimeNet and SpeedNet models. For each model, we discuss the data set the models are trained on, the models and the results of our experiments. First, we discuss if and how well the TimeNet models have learned time. Then, we discuss if and how well the SpeedNet models have learned velocity. For SpeedNet, we will also analyse the results to gain insight into the SpeedNet model performance and its limits.

5.1 Experiment 1: Measuring Time

We will use TimeNet to predict the time that has passed between frames. The model must learn what time looks like, by analysing the size of the displacement.

Given a stationary camera at the side of the road that records images at a fixed frame rate (FPS) of a vehicle driving by at a fixed speed. Between frames f_0 and f_1, the car will have moved over a certain distance. Since the distance to the vehicle and the speed of the vehicle are fixed, the model will be able to learn how much time has passed. If we keep the velocity of the vehicle the same, but increase the time between f_0 and f_1, the vehicle will have travelled a larger distance. If TimeNet can learn the relation between the magnitude of these displacements, it will have learned how much time has passed between the frames.

The observed object will have to move at the same pace and distance across the receptive field of the camera. If the object moves closer or changes pace, the displacements across the image will change. This will make it difficult for TimeNet to learn time from image pairs.

5.1.1 Data set

To learn the time between frames we use the KTH data set [29], the first action recognition data set. It is a small data set of 25 people performing six different tasks: walking, jogging, running, boxing, hand waving and hand clapping. The 25 people performed the actions in four different settings: outdoors, outdoors with scale variation, outdoors with different clothes and indoors. In all videos, the backgrounds are homogeneous. The videos are recorded using a static camera with an FPS of 25. On this data set, we will determine if FlowNet's base knowledge can be used to learn time from two consecutive images.

We have used the data set's train and test splits to conduct this experiment. For this experiment, we will train our models on the *walking* set and we will evaluate our models on both the walking and running sets. Since the settings in the videos differ, we only train and evaluate on the indoor setting. These settings contain people walking across the image at roughly the same distance from the camera, allowing the models to learn the time between frames. For this research, we have used the same videos of the train and test splits, but we have dropped frames where there is no person in the image.

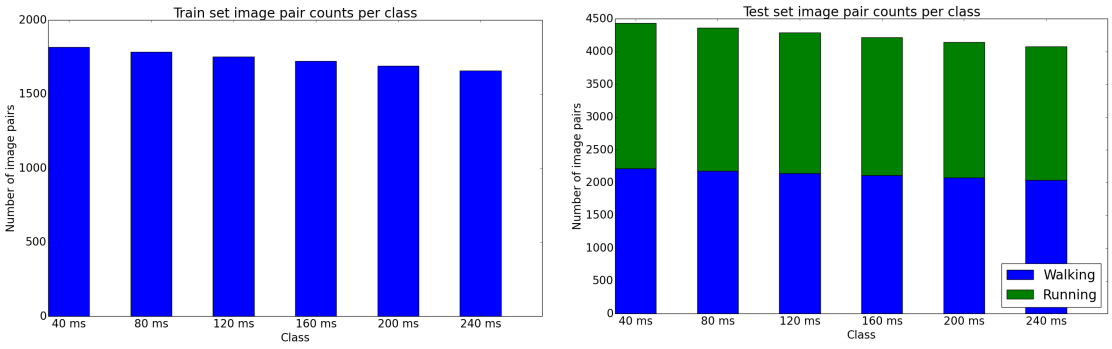


Figure 10: Train and test set statistics of the KTH data set. The horizontal axis displays the class, whereas the vertical axis displays the number of training pairs.

We did this to reduce the ambiguity in the data set. This has lead to a data set with a total of 1850 images.

5.1.2 Settings

As the videos of the KTH data set are recorded with an FPS of 25, the default time between frames f_0 and f_1 is 40 ms. When we skip frames, the time between frames increases. If we skip one frame and use f_0 and f_2 as a training pair, the time between frames is 80 ms. We have repeated the skipping of frames to create a data set with six different time periods between frames. This leads to the following classes: 40, 80, 120, 160, 200 and 240 ms. In Figure 10, it is shown how many unique image pairs we can make from this data set, per class. To train the models, we use 3000 image pairs per class, creating a total training set of 18000 image pairs. Given the number of image pairs per class, examples will be repeated in the training set.

To evaluate the model, we will use both the indoor walking and running data set. The faster a person moves, the larger the displacement. Similarly, the more time between frames, the larger the displacement. Since a lower FPS equals more time between frames, we hypothesize that if a person is running the models will predict a lower FPS.

5.1.3 Models

We have formulated this experiment as a classification problem. There is no knowledge of any other research that attempts to learn what time looks like, causing us to create our own baseline. Our baseline consists of a *support vector machine* (SVM). We compare the results to our TimeNet models.

Baseline We train an SVM from the standard Python package *scikit-learn* [23]. To train this model, the original FlowNet models were used. The features of KTH image pairs were extracted from the *conv6* layer from the FlowNetCorr and FlowNetSimple models. The *conv6* layer returns a 1024-dimensional vector, which are fed to the SVM.

		40 ms	80 ms	120 ms	140 ms	200 ms	240 ms	Average
SVM	[Corr]	0.16	0.17	0	0.14	0.47	0.15	0.18
SVM	[Simple]	0.33	0	0.06	0.37	0.3	0.35	0.24
TimeNet	[Corr]	0.31	0.15	0.11	0.15	0.21	0.32	0.21
TimeNet	[Simple]	0.70	0.58	0.47	0.36	0.35	0.57	0.51

Table 1: Precision per model and class. The last columns displays the average precision of the model.

During training time, we train two SVM models; one on the *conv6* feature vectors of FlowNetSimple and the other on the *conv6* feature vectors of FlowNetCorr. After testing we found that the best performance was found at a penalty parameter value of $C = 1$ for the vectors of both FlowNetSimple and FlowNetCorr. These models are used as comparison in this research and will be referred to as SVMSimple and SVMCorr, respectively.

TimeNet The models under test are the correlation and simple TimeNet models. We will refer to them as TimeNetCorr and TimeNetSimple. By means of experiments, we have come to find that the number of fully-connected layers does not improve the TimeNet model performance. Therefore, we have chosen to use one fully-connected layer in our TimeNet models.

5.1.4 Results

In this section we evaluate the TimeNet models. We focus on determining if our models can learn a relation between time and displacements. The models we evaluate are SVMCorr, SVMSimple, TimeNetCorr and TimeNetSimple. We will use this experiment to determine if we can use FlowNet as a base model to learn velocity. Our evaluation consists of:

- The performance of each model, per class
- An analysis of the TimeNet models' prediction strength

Performance To analyse the performance of our models, we evaluate our models on the walking test set. We created a confusion matrix of each model and calculated the precision and recall of each model. The precision can be found in Table 1. Table 2 displays the recall per model.

Table 1 displays the precision of each model, per class. We chose this metric as this indicates how often an predicted class is correct, which allows us to determine if the model we evaluate is biased. From Table 1, we see that no model performs well. The SVMCorr and TimeNetCorr models are both outperformed by the SVMSimple and TimeNetSimple models. TimeNetSimple outperforms all models considerably and appears to have learned time to a certain degree.

		40 ms	80 ms	120 ms	140 ms	200 ms	240 ms	Average
SVM	[Corr]	0.21	0.45	0	0.11	0.02	0.18	0.16
SVM	[Simple]	1	0	0.007	0.15	0.12	0.58	0.29
TimeNet	[Corr]	0.44	0.04	0.1	0.15	0.12	0.58	0.24
TimeNet	[Simple]	0.9	0.56	0.42	0.33	0.27	0.68	0.53

Table 2: Recall per model and class. The last columns displays the average recall of the model.

Analysing the separate precision per class, we can see that most scores are low. The low scores mean that the models either predict a specific value very often or very little. From this table, it appears as though SVMCorr is biased to predict a time of 200 ms, making this model highly unqualified to learn time. TimeNetCorr performs poorly as well, though the model does not appear to be as biased as SVMCorr. TimeNetCorr appears to mostly predict a very low or very high time between frames, while most likely makes wrong predictions. This is most likely caused by the correlation layer as this layer maps pixels between the two images within a specified neighbourhood. Since the mapping occurs on a pixel level and each person walks at its own speed, it is difficult for the Correlation models to learn time. The SVMSimple model performs better than the Correlation models, but still performs poorly with an average precision of 24%. It is possible that the Simple models perform better than the Correlation models, due to its freedom to learn time in a less restrictive manner. The correlation layer drives the Correlation models to take into account changes on a pixel level, where the Simple architecture allows for a more abstract internal representation. From the class precisions, we can see that the model hardly ever predicts a time of 80 ms and 120 ms. This could mean that the model has trouble distinguishing times from one another. The model that performs best is TimeNetSimple. TimeNetSimple performs best when predicting a time of 40 ms. However, the model performs worst when predicting 140 ms and 200 ms. This could mean that the displacements appear similar to the model, leading the model to make a wrong prediction.

In Table 2, we have displayed the recall per class. This helps us gain insight into how often the correct class was predicted. We again see that each model performs poorly, with TimeNetSimple performing best. The Correlation models perform worst. Though SVMCorr is biased, the times it predicts a time of 80 ms the model is correct almost half the time. This shows that, though SVMCorr is biased, it has successfully mapped some displacements to a time. The results from the precision indicated that TimeNetCorr is biased to predict either 40 ms or 80 ms. This is reflected in the results of Table 2. The highest recall is achieved in these classes, while predictions of all other classes are very low. SVMSimple performs excellent on predicting a time of 40 ms. The model performs very poorly when predicting 80 ms and 120 ms. The recall results show that this model is also biased to predict either 40 ms or 240 ms, which we is not reflected in the precision. TimeNetSimple again performs best. The prediction of little time between frames is quite good, compared to the results of all other models. When the time between frames

increases, it becomes more difficult for the model to determine the time between frames.

Analysis The TimeNetSimple model achieves a precision of 0.51 and a recall of 0.53. We hypothesize that it is possible for the TimeNet models to map a displacement to a value, but that the value the displacements are mapped influence the models' performance. For TimeNetSimple, we can see in Table 2 that the model has trouble predicting the times of 140 ms and 200 ms. Given the way we built our data set, we understand that ambiguity is introduced. Each person in the data set walks across the image in their own pace. This differs per person, already introducing a degree of ambiguity. When we create our data set to introduce different times between frames, this ambiguity increases in the same manner. For example, if a person walks slowly and we skip a frame to create a test pair for 80 ms, their movement could be equal to a person that walks quickly with a time of 40 ms. The more frames that are skipped, the more this should set in. If this is the case, TimeNetSimple can map displacements to time.

Running test set To confirm if TimeNetSimple can map displacements to time, we will evaluate the model on the running test set. Ambiguity is introduced in the data due to the walking speed of each person. By evaluating TimeNetSimple on the running data set, the displacements are larger by default. This should cause TimeNetSimple to predict a larger time. If this is the case, the same base model can be used to learn velocity.

To evaluate TimeNetSimple's performance on the running test set, we have created a confusion matrix that can be found in Figure 11. The labels are depicted horizontally and the predictions are displayed vertically. We see a gradual change in predictions. The prediction of 40 ms is oftentimes at a prediction of 80 ms. When people are running with a time between frames of 80 ms, the predictions are mostly divided over 160 ms, 200 ms and 240 ms. This confirms our theory that each person walks and runs at their own pace and that it introduces ambiguity in the data set. People running with a time between frames of 120 ms or higher is almost always predicted as a time between frames of 240 ms. It appears that displacements larger than TimeNetSimple was trained on are classified into the highest time between frames the model knows.

5.1.5 Findings

We have performed an experiment to determine if we can learn time from two consecutive images. We have shown that it is possible to learn time, but that this is a difficult problem to solve. For our models to learn time, we require a static camera that captures objects passing across the image at the same speed and distance from the camera. We have created a data set of people walking across the image and adhere a similar distances from the camera and movement across the camera. The models have trouble learning time from this data set, due to the fact that each person walks at their own pace. We have validated that this ambiguity does exist in the data set and that this leads to poor prediction.

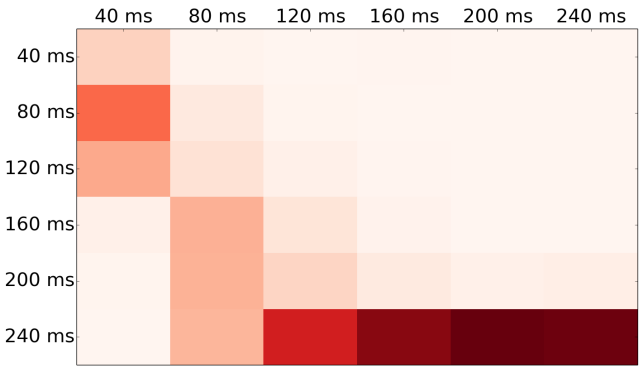


Figure 11: Heatmap of TimeNetSimple, evaluated on the *running* test set. The columns are the labels and the rows are the predicted values.

Tough the overall performance of all models is poor, we hypothesize that FlowNet can be used as a base model to learn velocity. We have shown that our model can map a displacement to time and if our model can do this on such an ambiguous data set, our model should be capable of mapping a displacement to a velocity. The data set to learn velocity will contain much less ambiguity, as the time factor is fixed. This loosens some restrictions on our data set, which makes the velocity problem easier to solve.

5.2 Experiment 2: Predicting Velocity

This experiment entails the core of our research; for the model to learn velocity. We will use SpeedNet to do this. SpeedNet must learn that the magnitude of each displacement can be different over the same time span and learn that this indicates speed. If the model is successful to map the size of a displacement to a speed, the model has learned what speed looks like.

Given a stationary camera mounted on a car that records images at a fixed FPS, the camera will capture images while the car drives around. By looking at all moving parts in the image, SpeedNet must determine how fast the car is driving. This is a different problem to learning time, since each pixel in the image will move and there are no fixed speeds. Additionally, there is other traffic, pedestrians and buildings that could make it difficult for the model to generalise well. However, if the model is successful in building an internal representation of each speed it will be able to predict speed.

5.2.1 Data set

The KITTI data set [7] was created as a benchmark for stereo flow, optical flow, visual odometry, 3D object recognition and 3D tracking. The data was created by equipping a car with static high-resolution colour and grayscale cameras and driving through streets of Karlsruhe, Germany. The ground truth was collected by using a Velodyne laser and

GPS, where the speed of the vehicle is can be calculated to meters per second (m/s). During the trips, videos were made, while also tracking the speed of the car. The data set has been processed by the authors of [7], so that ground truth information is available about the number and location of other cars and pedestrians and more. To accommodate as many researchers as possible, the data set has been split into specific benchmarks.

Aside from the benchmarks, it is also possible to download two different raw data sets of the trips; synchronised and unsynchronised. The camera on the vehicle records at a frame rate of 10, whereas the GPS records information much more often, leading to an unsynchronised data set. For this research, we will use the synchronised raw data set, as each frame has been linked to the vehicle GPS information at that time by the authors.

The KITTI raw data set does not have train or test splits, so we have made these ourselves. The raw data set consists of trips in residential areas, the city, roads outside the city and on a campus, along with GPS data per frame. There are also categories in person recognition and tracking, which can be used for calibration purposes. These two categories were not used during this research. The videos that were used were split into a train and test set. Videos that were similar, such as standing still at a traffic light, were split evenly into the train and test set. The actual splits can be found in Appendix A.

5.2.2 Settings

The images in this data set are quite large. Though the size of the images vary somewhat, the general size of the image is 1230×370 pixels. This is a large image and, along with the complexity of the models, takes quite some time to train. Therefore, the center of the images were extracted to train the models on. This lead to a train image size of 610×370 pixels. The center of the image should contain smaller displacements than in the peripheral of the image. We hypothesize that the model can use this information to learn speed better, since the original models were trained on small displacements.

The speeds that are used to train on vary between 0 and 18 meters per second (m/s). Though the KITTI data set does contain higher speeds, most data could be found in this speed range. To train the models, we collected 1000 image pairs of each speed, to create a balanced data set of 18000 training pairs. In this case, it is certain that some pairs will be used repeatedly to train on within an epoch. However, we chose this approach to keep the training set as large and balanced as possible. Statistics on how the speed is divided over the train and test set can be found in Figure 12.

5.2.3 Models

As speed is continuous, we use regression for this experiment. There is no other research known that attempts to learn speed, causing us to create our own baseline. Our baseline consists of a standard LASSO model [33]. We compare the results to the SpeedNet models that we have trained.

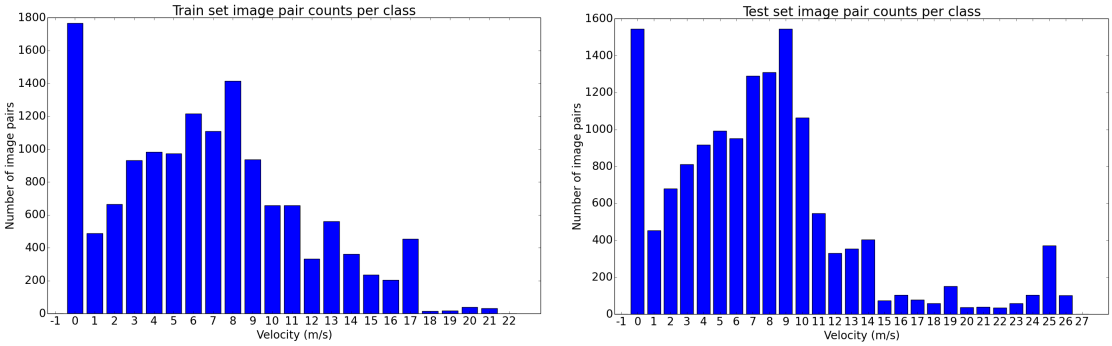


Figure 12: Train and test set statistics. The horizontal axis displays the speed, whereas the vertical axis displays the number of training pairs.

Baseline We train a linear regressor, LASSO, from the standard Python package *scikit-learn* [23]. To train this model, the original FlowNet models were used. The features of training pairs were again extracted from the conv6 layer, which is a 1024 -dimensional vector in both FlowNetSimple and FlowNetCorr. Along with this vector, the speed of the vehicle is passed. We will train two LASSO models; one on the *conv6* vectors of FlowNetSimple and the other on the *conv6* vectors of FlowNetCorr. We name the trained models LassoSimple and LassoCorr, respectively. In the case of LassoSimple, the regularization multiplier is set to 1^{-5}. For LassoCorr, the regularization multiplier is set to 1^{-6}. Though the regularization parameter could contain many different values, we have come to find the best performance of these LASSO models with these settings after experimentation.

SpeedNet The models under test are the correlation and simple SpeedNet models. We will refer to them as SpeedNetCorr and SpeedNetSimple. By means of experiments, we have come to find that the number of fully-connected layers does not improve the SpeedNet model performance. Therefore, we have chosen to use one fully-connected layer in our SpeedNet models.

5.2.4 Results

In this section we evaluate our models that have learned speed. The models we evaluate are LassoCorr, LassoSimple, SpeedNetCorr and SpeedNetSimple. Our evaluation consists of analysing:

- the prediction error of each model, per speed
- the prediction during a full trip
- separate cases of good and bad predictions

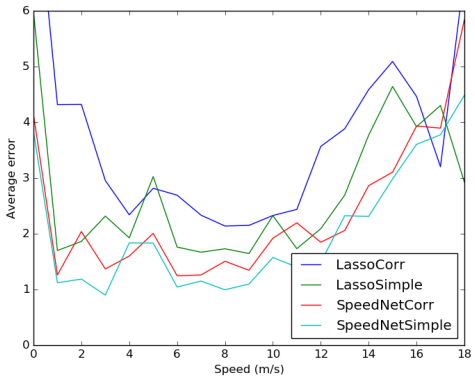


Figure 13: The average error of each model, per driven speed in m/s.

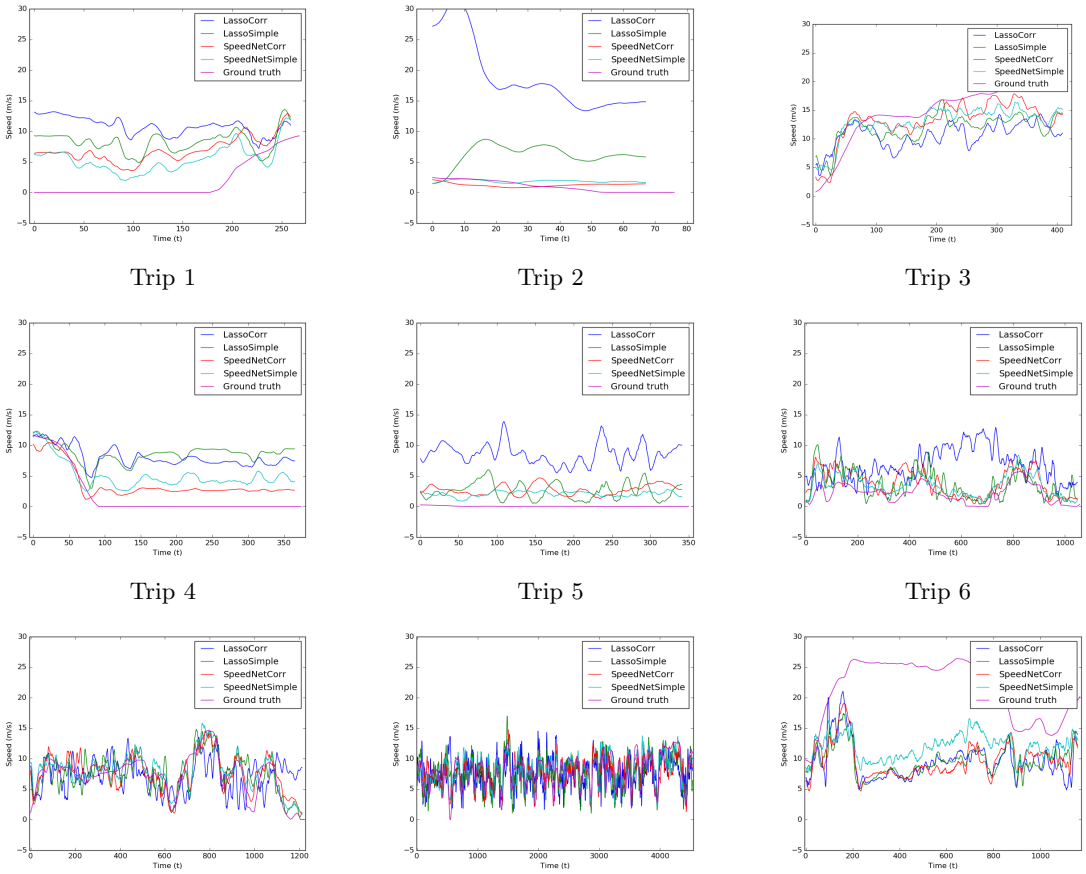
Average error To gain insight into our models' performance, we track their average prediction error, which is displayed in Table 3. It is shown that SpeedNetSimple has the smallest average prediction error over the entire test set, whereas LassoCorr has the largest average prediction error. Between the Correlation and Simple models, the Simple models outperform the Correlation models.

Model	Average error
LassoCorr	3.75
LassoSimple	2.74
SpeedNetCorr	2.39
SpeedNetSimple	2.05

Table 3: Average error per measured velocity in m/s, per model.

To gain more insight into the models' prediction power, we have plotted the average error per speed in Figure 13. We can see that the models predict low speeds well. Though we expected LassoCorr and SpeedNetCorr to perform best due to the correlation layer, the models are outperformed by LassoSimple and SpeedNetSimple. However, the models perform significantly worse as the speed of a vehicle increases. In the case of SpeedNetCorr and LassoCorr, the displacement in the correlation layer is set too low. For each pixel, the Correlation model will attempt to match this pixel in a specified neighbourhood. However, if the pixel falls outside this neighbourhood the pixel cannot be mapped. This will cause the model to miss information, which leads to poor speed predictions.

The models that perform best overall are the SpeedNet models. Interestingly enough, SpeedNetCorr should be optimized for small displacements, which occur at low speeds. However, the model is almost always outperformed by SpeedNetSimple. Therefore, we hypothesize the underlying FlowNetSimple model can learn optical flow well enough to learn speed, but not well enough to map optical flow properly on a pixel-by-pixel level.



Trip 1 Trip 2 Trip 3

Trip 4 Trip 5 Trip 6

Trip 7 Trip 8 Trip 9

Figure 14: Several car trips our trained models have been tested on. The figures show how fast the car was driving at which time in the video and the speed that each model predicted. For plot purposes, the values predicted by the models are a moving average of 10 values.

It appears as though this imperfection makes FlowNetSimple the better model to learn velocity, as the model knows displacements and that information can be adjusted better to learn speed compared to FlowNetCorr.

The best speed prediction still contains an average error of 1 m/s. Though an error in predictions is expected, it is not expected to be this large. In the following experiments, we will attempt to understand why this error occurs.

Prediction of trips To gain more insight into the behaviour of the models and their performance, we have let the models predict the velocity over different trips. The results are plotted in Figure 14. As can be seen in Figure 13, the lowest average error prediction of speed is about 1 m/s. Even when the best model makes a good prediction, chances

are that the prediction is off. By analysing the prediction, we have smoothed our plots. We plot a moving average of the predictions in Figure 14. The moving average uses a window of 10 values (1 second) to determine the speed prediction. Smoothing our plots results in less clutter and allows us to analyse the performance of our models.

Each model has trouble predicting very low speeds or when the vehicle is standing still, as can be seen in Trips 1, 2 and 4. Though the SpeedNet models come closest to correct predictions, the Lasso models are both most often wrong when predicting low speeds. Given the way SpeedNetCorr was designed, it was expected that this model would be good at predicting low speeds. This is not always the case as SpeedNetSimple sometimes outperforms SpeedNetCorr.

With higher speeds, between 5 m/s and 15 m/s, all models perform much better. Though the predictions are off at times, the trends of the trips are quite accurate. In Trip 3, all models predict an increase in speed that, until the speed becomes higher than 18 m/s, is followed by the models. The same behaviour can be seen in Trips 6, 7 and 8. Trip 8 is long and difficult to analyse. Therefore, this was added in greater detail in Appendix B.

The models were trained from speeds of 0 m/s to speeds of 18 m/s. In trip 9, the driving speed becomes much higher than the models are trained on. A noticeable dip in speed predictions occurs. Instead of the models predicting the highest speed they were trained on, most make a prediction of speeds between 5-10 m/s. SpeedNetSimple makes a prediction of about 11.6 m/s, which slowly increases, but never reaches the true speed of the vehicle. When the speed later drops back down to a speed the models were trained on, the prediction is poor.

When analysing the trends in Figure 14, we determine that the models are poor at predicting low and high speeds. Speeds between 5 m/s and 15 m/s are predicted much better, albeit more of a trend that approaches the speed of the vehicle than predicting the actual vehicle speed.

Analysis To understand why the SpeedNet models behave as they do, we take a closer look at the data and the results. In the case of low speeds, the SpeedNet models perform poorly. The situations that we will analyse can be found in Figure 15. The first two images are the input images. The third image shows the two images blended together, whereas the last image displays the two images subtracted from each other.

The images in the first row are from Trip 1, where we take a closer look at where the vehicle stands still. When blending the images together, we see that over half the image looks clear. On the right, the moving vehicles blend together. Looking at the fourth image of the first row, we only see the part of the image that moves. Since the model is trained to learn optical flow, it will have created an internal representation similar to the fourth image where there is only movement on the side of the image. Since optical flow focuses on movement, it is possible that our SpeedNet models mainly gain information from moving parts in an image. This makes it difficult for the SpeedNet models to predict that the vehicle is standing still when parts of the image are moving.

The second row comes from Trip 2. At this point, the vehicle is slowly rolling as it

waits in front of a red traffic light. Again, there is movement on the left and right of the image. However, this movement is much smaller than in the top example, as can be seen in the third image of the second row. The subtracted image displays less areas in the image that are not moving, but the prediction of this speed is better than the speed of Trip 1. It appears as though any kind of movement in the image is taken into account when calculating speed and that the static parts of the images are mostly disregarded. Since optical flow is an approach to tracking moving objects, it is possible that static areas in images are deemed unnecessary or not interesting. Given our training set, it is also possible that lack of movement is heavily under represented, leading our models to focus on extracting velocity information from moving parts of the image.

The third and fourth row show images from Trips 7 and 8. The images we analyse are both taken from a speed between 5 and 10 m/s. As emphasized in Appendix B, both SpeedNet models predict the speed quite well. The trend of the vehicle speed is largely followed if the prediction is wrong. Where we see that SpeedNetCorr tends to overshoot in its predictions, SpeedNetSimple makes more accurate predictions, unless the speed of the vehicle drops very low. This can be seen at times $t \approx 600$, $t \approx 1500$ and $t \approx 2300$ in Trip 8 of Figure 14.

In Trip 9, the model makes unexpected predictions. All models were trained to predict speeds until 18 m/s. During Trip 9, a speed of approximately 25 m/s is achieved. However, when trying to predict such a high speed, each model drops in its speed prediction. We take a closer look at the prediction the model makes at $t = 600$. In the fifth row of Figure 15, we see in the third image that the white lines on the asphalt become less visible and overlapping each other slightly. In the fourth image, the lines are clearly visible, and where the lines overlap is disregarded. Additionally, the setting is monotonous and lacks context compared to the other trips. The asphalt itself and the sky take up the largest part of the image. This leaves the model to determine optical flow from the lines on the asphalt, the bush on the left and the traffic sign on the right. As the model has little context to determine the velocity from, the predictions suffer.

To gain insight into the importance of context, we have conducted a small experiment. We have drawn the lines on the asphalt and prevent them from changing in the images. The resulting two images were then passed to the model again and the results were collected. The images are displayed at the bottom row of Figure 14. As can be seen in the third image, the lines are not shown double. Therefore, the speed prediction will no longer depend on the changes of the lines. For these two images, the predicted velocity was originally 8.97 m/s by the SpeedNetCorr and 11.63 m/s by SpeedNetSimple. The prediction after augmentation of the images leads to predictions of 12.51 m/s and 12.45 m/s by SpeedNetCorr and SpeedNetSimple, respectively. This shows that the lines on the asphalt are taken into account when determining speed for both SpeedNet models. Given the repetition of the lines on the asphalt, ambiguity is introduced. The ambiguity creates the illusion that the vehicle is moving at a much slower pace, which leads to poor predictions of velocity.

The augmentation of the data has the largest effect on the prediction of SpeedNetCorr. Since SpeedNetCorr is optimized to predict small displacements due to the

Model	10 m/s	20 m/s	24 m/s
SpeedNetCorr	9.27	12.54	11.35
SpeedNetSimple	9.48	13.76	13.68

Table 4: The predictions made by the SpeedNet models on our created speed data. Each speed was created by skipping a frame. To create the speeds of 10 m/s, we used frames at times t and $t = t + 2$ when the vehicle moved at a speed of 5 m/s. The same was repeated with 10 m/s and 12 m/s to create the speeds of 20 m/s and 24 m/s, respectively.

correlation layer, taking away the ambiguity of the lines on the asphalt leads to better predictions. However, the improvement in predictions is much less for the SpeedNet-Simple model. The SpeedNet model has learned what velocity looks like by taking into account all displacements that can be detected between images. By removing the ambiguity of the lines on the road, the predicted speed goes up. These results show that the lack of context and repetitive factors on and beside the roads make it difficult to determine the speed of the vehicle on a highway. Taking away the lines on the road, very little context remains for the model to determine speed, which appears to also negatively affect the performance of the SpeedNet models.

Context experiment During analysis of Trip 9, we determined that it is possible that the lack of context could impact the speed prediction. To analyse if this is the case, we have set up a context experiment. During this experiment we choose pieces of trips with more context than Trip 9, where the vehicle travels at a speed of approximately 12 m/s. As in Experiment 1, we will skip a frame. This will increase the displacement of the objects in the image, appearing as if the vehicle is moving with a speed of 24 m/s. To validate this experiment, we will perform the skipping of frames with speeds of 5, 10 and 12 m/s which in turn become a speed of 10 m/s, 20 m/s and 24 m/s. The experiments were performed 10 times and the averaged results are displayed in Table 4. Images used in this experiment are displayed in Figure 16.

Table 4 displays the results per calculated speed. We created a speed of 10 m/s by having the SpeedNet models predict speed on images taken at a speed of ≈ 5 m/s at times t and $t = t + 2$. This generates a speed of 10 m/s and shows to work well, given the average predictions made. The average predictions are a speed of 9.27 m/s by SpeedNetCorr and 9.48 m/s by SpeedNetSimple. Given the results in Figure 13, this is to be expected. Therefore, we can perform the trick of skipping frames to create situations where the model has to predict speeds it was not trained on.

We repeat the skipping of frames to create speeds of 20 m/s and 24 m/s from images with more context. We see that the predictions of higher speeds are still poor. The extra context allows the models to make a better prediction, but that the models still have problems generalising its prediction to velocities they have not been trained on. It is possible that there is another factor in the data that introduces ambiguity or makes it more difficult to determine high velocity from images.

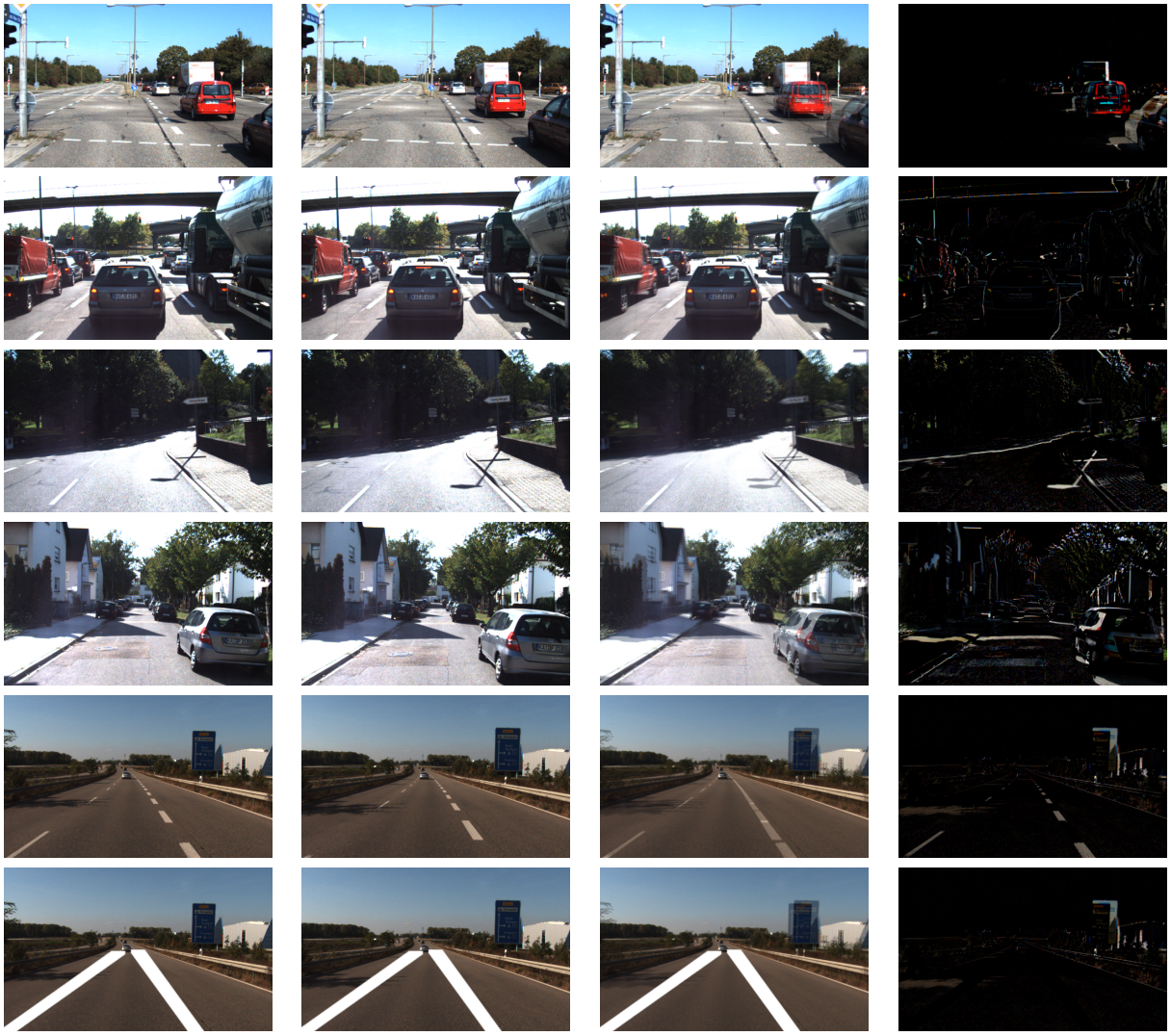


Figure 15: The first two images are the consecutive images passed to the SpeedNet models. The third image displays the images blended together and the last image displays the two images subtracted from each other. Each row displays a situation from a trip we analyse. From top to bottom, we display: Trip 1 at $t = 100 - 101$, Trip 2 at $t = 20 - 21$, Trip 7 at $t = 700 - 701$, Trip 8 at $t = 1000 - 1001$, Trip 9 at $t = 600 - 601$ and an edited version of Trip 9 at $t = 600 - 601$.

Findings Overall, SpeedNetSimple outperforms SpeedNetCorr. The correlation layer drives SpeedNetCorr to consider displacements on a pixel level within a specified neighbourhood. If the displacement is large, the pixel cannot be mapped, which restricts SpeedNetCorr. Additionally, it is possible that the pixel-level mappings cause SpeedNetCorr difficulty in learning velocity. The distances on pixel level can differ, making it

Figure 16: A figure displaying the images used to test how much context influences the speed prediction. The top row consists of images at a calculated speed of 10 m/s, the second row 20 m/s and the bottom row 24 m/s.

difficult for the model to generalise displacements to a velocity. Since SpeedNetSimple isn't restricted by the correlation layer, the model has the freedom to learn velocity without restrictions which leads to better performance on prediction of higher speeds. The predictions of SpeedNet also suffer from the depth of the base model. Given the fully-convolutional architecture of the base model, the models should be able to learn speed better if the architecture is deep enough. Overall, the model can predict speeds until 15 m/s if we smooth the results. The trends are followed by SpeedNet models very well. We expect better SpeedNet performance by further optimizing FlowNet.

We find that the SpeedNet models depend heavily on context to predict velocity. If the vehicle stands still and the models detect movement in the image, that information is used to predict speed. When the vehicle moves with speeds of 5 m/s to 15 m/s, the environment often contains buildings or parked cars that can be used to determine the speed of the vehicle. The predictions become poor as the velocity of the vehicle increases. When the vehicle moves with high velocity, the environment is often monotonous and repetitive which leaves our SpeedNet models with little context to determine speed with. This leaves the model to heavily depend on little context to determine speed. It has been shown that the lines on the road will overlap, creating bias towards a low speed prediction. We have attempted to add context to high-speed displacements by skipping frames. This has shown that more context does lead to an improvement in prediction, but that the prediction is still poor for high speeds.

6 Discussion

During the evaluation of TimeNet, we have come to find that it is extremely difficult to learn time from consecutive images. In our case, it is imperative that all movement occurs at the same pace and at the same distance from the camera. In the KTH data set, this is not the case. There are slight differences in the image as a person walks faster than the other or from a slightly different distance. This creates ambiguity in the data and makes it difficult to learn time. However, the TimeNetSimple model has shown to be capable to learn the relation between time and the magnitude of a displacement best. During evaluation, we have come to find that the size of the displacement can indeed indicate how much time has passed between frames making FlowNet a proper base model to learn velocity from. However, the results of this research will most likely not be used in reality, as these data restrictions hardly ever occur naturally.

When analysing the results of TimeNetSimple, we have come to find that the model predicts a time of 240 ms if the displacement is larger than the model was trained on. If the model knows displacements, it is possible that the 240 ms class has become a class where each displacement of 240 ms and larger is assigned to. However, we will have to conduct more research into this behaviour.

During the evaluation of SpeedNet we found that the model does not always detect that the vehicle is standing still. An example of this is in the top row of Figure 15. If there is movement in the image, SpeedNet predicts the vehicle is moving. This is caused by the fact that the base model, FlowNet, is trained to detect optical flow. The goal of optical flow is to detect changes in images. Displacements between the images appear to contribute more to the prediction of speed than the lack of movement between the images. This is necessary to successfully predict optical flow, but performs poorly when predicting velocity. This appears to be inherited knowledge of the base model that proves to negatively influence our model's prediction at low velocities. When predicting velocity, both movement and the lack thereof should contribute to the velocity equally.

When the vehicle travels with a high velocity, SpeedNet predicts a lower velocity. Though we initially expected the model to predict the highest velocity the model was trained on, this is not the case. We hypothesize that this is caused by combination of several factors. First, high velocity is reached on highways. These roads contain little context for SpeedNet to determine its speed off. The context the model can use are road signs, the lines on the road and other buildings or plants at the side of the road. However, any context at the side of the road are few and far in between. This leaves SpeedNet to mainly base its speed off the lines on the road. However, the lines on the road are repetitive. When travelling at high velocity, the stripes on the road will overlap, creating the illusion of the vehicle travelling at a low velocity. This leaves the model to predict a speed from little context, which leads the poor results.

The SpeedNet model predicts velocity well, as long as the ego-motion of the vehicle is 5 to 15 m/s. Given the insight we gain from SpeedNet's performance at low and high speeds, we see that vehicles that travel at within this velocity interval tend to find themselves in situations where there is enough context to predict the velocity of the

vehicle. These speeds are achieved in cities or traffic jams, where buildings, cars and signs are available to use as context. We have come to find that there is high variance in the prediction per image pair. However, when we smooth the predictions over 10 samples the results follow the trend of speed accurately. Predicting the actual speed of the vehicle remains a difficult task, but the smoothed predictions are accurate.

Given the performance of TimeNetCorr, we expected SpeedNetSimple to heavily outperform SpeedNetCorr. However, our experiments have shown that this is not the case. Though SpeedNetSimple performs best overall, SpeedNetCorr also performs well despite its limits introduced by its correlation layer. This shows that the Correlation model can successfully map a displacement to a value, but has trouble with predicting time. However, it is unclear why TimeNetCorr has these limits. That this occurs due to a combination of the correlation layer and the ambiguity in the data set. The ambiguity in the data set could make it difficult for the Correlation models to predict large times between frames. Addtionally, the ambiguity in the data could make it difficult for the Correlation models to determine a low time between frames. Understanding why the Correlation models and features yield such poor time predictions will require more research. Since this research focused on learning velocity, we have not researched this behaviour.

7 Conclusion

The purpose of this thesis was to determine if we can learn speed from two consecutive images. We have proposed the theory that optical flow can be used to learn what velocity looks like. We have discussed the base model, FlowNet, and our own models, TimeNet and SpeedNet in depth in Section 4. We have evaluated the models and their limitations in Section 5. Our findings were discussed in Section 6.

We have used transfer learning techniques to alter the architecture of the FlowNet model and create two SpeedNet and TimeNet models; a Simple model and a Correlation model. To create our models, we reuse the trained layers from FlowNet that determine optical flow, to exploit their knowledge. We mildly retrain the layers during training time to adjust the model to the new data set and problem. This has led to models capable of predicting time and velocity.

We have discovered that predicting time from images is very difficult. The data restrictions specify that movement should occur at a certain pace and a certain distance from the camera. Even if training were successful, prediction of time would only be successful for this specific pace and distance from the camera. Though this shows that TimeNetSimple cannot be used in real life, the model has shown capable of learning a relation between the magnitude of a displacement and a time. This proved that our model should be able to learn velocity.

When predicting speed, we have come to find that context is very important. If there is no context, neither SpeedNet models can track the displacements of pixels and make a velocity prediction. With low speeds, there is often enough context as the vehicle is most likely at a stop light or in a street. On the highways, however, there is significantly less context from which the model can determine its velocity. In this case, each bit of context weights heavily to determine the speed of a vehicle. The lines on the road, which are repetitive, introduce ambiguity. Due to the higher speeds, it appears as though the lines on the road have hardly changed although this is not the case, which causes the model to predict low velocity.

Another problem the SpeedNet models introduce is their reliance on movement. Since the base models are trained to detect optical flow, any kind of movement holds weight over the lack of movement. This is correct when predicting optical flow, but creates a problem when predicting velocity. When predicting velocity, the lack of movement is just as important. This leads to poor predictions at a stop light when surrounding traffic is moving.

Though both SpeedNet models perform well, SpeedNetSimple outperforms SpeedNetCorr. SpeedNetSimple has a generic architecture, whereas SpeedNetCorr was derived from an architecture to learn optical flow. We determine that SpeedNetCorr has been trained to learn velocity from pixel displacements, where the SpeedNetSimple architecture remains free to learn velocity in a less strict manner. Both models suffer from the ambiguity in the data. However, the Correlation layer in SpeedNetCorr will remain a restriction when learning velocity. Therefore, a more generic approach performs better and state that SpeedNetSimple is the better model to predict velocity.

In the velocity range of 5 to 15 m/s, SpeedNet performs well in predicting the trend of speed. We have smoothed the predictions, which shows that both SpeedNet models are good at predicting the trend of the speed. It does not often happen that the velocity prediction is exactly as labelled, but using a moving average improves the predictions significantly. That the images within this velocity interval contain enough context for the model to reliably predict velocity. There is enough movement to not be biased by the movement of other vehicles and there is enough context for the model to correctly predict a speed. With these results we can conclude that it is possible to learn velocity, however ambiguity remains when predicting high speeds.

8 Future work

There are several ways this model can be improved. In this section we discuss the changes and improvements that can be made to this model to learn velocity better.

It is possible to perform the velocity prediction on different parts of the image separately, for the model to then determine speed itself. Currently, the model uses the entire image to determine velocity. If spatial pyramids are used to segment the image, the model can explicitly learn which parts of the image contribute to the speed most. The context of each segment can be preserved, allowing the model to determine the relation between the segments. This can help the model determine if it is basing velocity on the lines on the road or signs next to the road. As we have seen that the lines on the road contribute heavily to incorrect predictions, the patches of road can be forced to have a lower contribution to the overall speed prediction.

It is difficult for SpeedNet to determine if the vehicle is standing still if there is any movement in the image. There appears to be a bias towards any kind of movement in the model. This is to be expected as optical flow is the detection of movement. However, for the model to be so sensitive to movement negatively influences its performance. At low speeds, SpeedNet should detect the lack of movement just as much as it has to detect movement. We hypothesize that adjusting the layers in SpeedNet more will lead to better performance. By retraining the existing layers in the model, the performance of SpeedNet should increase. Currently, the adjustment we make to the layers in SpeedNet are quite small. They are effective, but could be more effective if adjusted more. We could experiment with the same learning rate and learning rate decay, while increasing the learning rate of the individual layers. For the SpeedNetCorr, the learning rate of the layers before the concatenation should remain small as these are only used to create a meaningful representation of each image. These changes could lead to a model that understands velocity better, allowing the model to also make correct predictions when standing still.

To plot the predictions of the trips, we have used smoothing. The predictions, as a sequence, contain high variance. We hypothesize that the variance in predictions will become less if we take into account multiple images to determine speed from. By using the SpeedNet models as they are, we could extract features from the last fully-connected layer and use those features to create a prediction over a larger window in time. Currently, we use features from images at $t = 0$ and $t = 1$ to predict velocity. If we were to increase this window and extract features between $t = 1$ and $t = 2$ as well, we create a smoothing function within our network. We will need to experiment with the size of the window we take into account when predicting velocity, but the predictions should contain less variance than they do now.

SpeedNet has problems predicting high velocity. This is partly due to the fact that FlowNet is a proof of concept. Though FlowNet reached good results, it is flawed. The authors have recently released a paper on FlowNet 2.0 [9]. This version of FlowNet consists of several stacked FlowNets, allowing the resulting model to predict very small and large displacements as well. Given the improved performance on large displacements,

FlowNet 2.0 is an interesting option to improve SpeedNet. However, in this research we have seen that the optical flow knowledge can lead to poor predictions, due to its bias toward any kind of movement. Therefore, the application of FlowNet 2.0 would still have to be combined with spatial pyramids and layer retraining to achieve better performance.

Currently, the velocity predictions heavily depend on displacements. However, the SpeedNet models could benefit from additional environmental information. This would allow the models to learn about speeds in high environments and learn which sections in the image give most information to predict velocity.

A Appendix A

Train set

2011 09 26 drive 0001	2011 09 26 drive 0036	2011 09 26 drive 0104
2011 09 26 drive 0002	2011 09 26 drive 0039	2011 09 26 drive 0106
2011 09 26 drive 0005	2011 09 26 drive 0046	2011 09 26 drive 0113
2011 09 26 drive 0009	2011 09 26 drive 0051	2011 09 28 drive 0016
2011 09 26 drive 0011	2011 09 26 drive 0057	2011 09 28 drive 0021
2011 09 26 drive 0013	2011 09 26 drive 0059	2011 09 28 drive 0034
2011 09 26 drive 0014	2011 09 26 drive 0060	2011 09 28 drive 0035
2011 09 26 drive 0015	2011 09 26 drive 0061	2011 09 28 drive 0037
2011 09 26 drive 0017	2011 09 26 drive 0064	2011 09 28 drive 0043
2011 09 26 drive 0019	2011 09 26 drive 0079	2011 09 28 drive 0045
2011 09 26 drive 0020	2011 09 26 drive 0084	2011 09 28 drive 0047
2011 09 26 drive 0022	2011 09 26 drive 0086	2011 09 29 drive 0004
2011 09 26 drive 0023	2011 09 26 drive 0087	2011 09 30 drive 0016
2011 09 26 drive 0027	2011 09 26 drive 0091	2011 09 30 drive 0020
2011 09 26 drive 0028	2011 09 26 drive 0093	2011 09 30 drive 0027
2011 09 26 drive 0029	2011 09 26 drive 0095	2011 09 30 drive 0028
2011 09 26 drive 0032	2011 09 26 drive 0096	2011 09 30 drive 0033
2011 09 26 drive 0035	2011 09 26 drive 0101	

Test set

2011 09 26 drive 0018	2011 09 29 drive 0026
2011 09 26 drive 0056	2011 09 29 drive 0071
2011 09 26 drive 0117	2011 09 30 drive 0018
2011 09 26 drive 0070	2011 09 30 drive 0034
2011 09 28 drive 0001	2011 10 03 drive 0027
2011 09 28 drive 0002	2011 10 03 drive 0034
2011 09 28 drive 0038	2011 10 03 drive 0042
2011 09 28 drive 0039	2011 10 03 drive 0047

B Appendix B

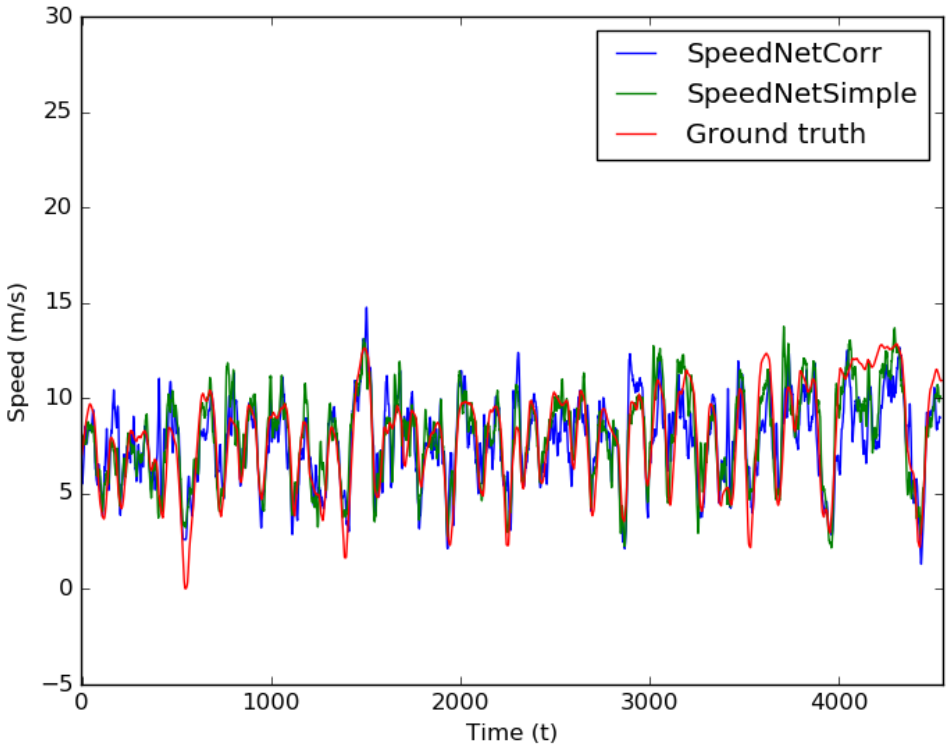


Figure 17: A plot of how well Trip 8 was predicted by the SpeedNet models. The predictions are a moving average of 10 values.

References

[1] Y-Lan Boureau, Jean Ponce, and Yann LeCun. A theoretical analysis of feature pooling in visual recognition. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 111–118, 2010.

[2] Thomas Brox and Jitendra Malik. Large displacement optical flow: descriptor matching in variational motion estimation. *IEEE transactions on pattern analysis and machine intelligence*, 33(3):500–513, 2011.

[3] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. A naturalistic open source movie for optical flow evaluation. In *European Conference on Computer Vision*, pages 611–625. Springer, 2012.

[4] Brian Chu, Vashisht Madhavan, Oscar Beijbom, Judy Hoffman, and Trevor Darrell. Best practices for fine-tuning visual classifiers to new domains. In *Computer Vision–ECCV 2016 Workshops*, pages 435–442. Springer, 2016.

[5] Jifeng Dai, Kaiming He, Yi Li, Shaoqing Ren, and Jian Sun. Instance-sensitive fully convolutional networks. In *European Conference on Computer Vision*, pages 534–549. Springer, 2016.

[6] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazırbaş, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *arXiv preprint arXiv:1504.06852*, 2015.

[7] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013.

[8] James J Gibson. The perception of the visual world. 1950.

[9] Eddy Ilg, Nikolaus Mayer, Tonmoy Saikia, Margret Keuper, Alexey Dosovitskiy, and Thomas Brox. Flownet 2.0: Evolution of optical flow estimation with deep networks. *arXiv preprint arXiv:1612.01925*, 2016.

[10] Shyr-Long Jeng, Wei-Hua Chieng, and Hsiang-Pin Lu. Estimating speed using a side-looking single-radar vehicle detector. *IEEE Transactions on Intelligent Transportation Systems*, 15(2):607–614, 2014.

[11] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd ACM international conference on Multimedia*, pages 675–678. ACM, 2014.

[12] Justin Johnson, Andrej Karpathy, and Li Fei-Fei. Densecap: Fully convolutional localization networks for dense captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4565–4574, 2016.

[13] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[14] Jinhui Lan, Jian Li, Guangda Hu, Bin Ran, and Ling Wang. Vehicle speed measurement based on gray constraint optical flow algorithm. *Optik-International Journal for Light and Electron Optics*, 125(1):289–295, 2014.

[15] Honglak Lee, Roger Grosse, Rajesh Ranganath, and Andrew Y Ng. Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations. In *Proceedings of the 26th annual international conference on machine learning*, pages 609–616. ACM, 2009.

[16] Stéphanie Lefèvre, Chao Sun, Ruzena Bajcsy, and Christian Laugier. Comparison of parametric and non-parametric approaches for vehicle speed prediction. In *American Control Conference (ACC), 2014*, pages 3494–3499. IEEE, 2014.

[17] Yi Li, Kaiming He, Jian Sun, et al. R-fcn: Object detection via region-based fully convolutional networks. In *Advances in Neural Information Processing Systems*, pages 379–387, 2016.

[18] M Litzenberger, B Kohn, AN Belbachir, N Donath, G Gritsch, H Garn, C Posch, and S Schraml. Estimation of vehicle speed based on asynchronous data from a silicon retina optical sensor. In *Intelligent Transportation Systems Conference, 2006. ITSC'06. IEEE*, pages 653–658. IEEE, 2006.

[19] Ce Liu, Jenny Yuen, and Antonio Torralba. Sift flow: Dense correspondence across scenes and its applications. *IEEE transactions on pattern analysis and machine intelligence*, 33(5):978–994, 2011.

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[21] Etienne Mémin and Patrick Pérez. Dense estimation and object-based segmentation of the optical flow with robust techniques. *IEEE Transactions on Image Processing*, 7(5):703–719, 1998.

[22] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Transactions on knowledge and data engineering*, 22(10):1345–1359, 2010.

[23] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[24] Xu Qimin, Li Xu, Wu Mingming, Li Bin, and Song Xianghui. A methodology of vehicle speed estimation based on optical flow. In *Service Operations and Logistics, and Informatics (SOLI), 2014 IEEE International Conference on*, pages 33–37. IEEE, 2014.

[25] Jerome Revaud, Philippe Weinzaepfel, Zaid Harchaoui, and Cordelia Schmid. Epicflow: Edge-preserving interpolation of correspondences for optical flow. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1164–1172, 2015.

[26] Dan Rosenbaum, Daniel Zoran, and Yair Weiss. Learning the local statistics of optical flow. In *Advances in Neural Information Processing Systems*, pages 2373–2381, 2013.

[27] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. In *Interspeech*, pages 338–342, 2014.

[28] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. *Artificial Neural Networks–ICANN 2010*, pages 92–101, 2010.

[29] Christian Schuldt, Ivan Laptev, and Barbara Caputo. Recognizing human actions: a local svm approach. In *Pattern Recognition, 2004. ICPR 2004. Proceedings of the 17th International Conference on*, volume 3, pages 32–36. IEEE, 2004.

[30] Deqing Sun, Stefan Roth, J.P. Lewis, and Michael J. Black. Learning optical flow. In *European Conference on Computer Vision*, 2008.

[31] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[32] Saber Taghvaeeyan and Rajesh Rajamani. Portable roadside sensors for vehicle counting, classification, and speed measurement. *IEEE Transactions on Intelligent Transportation Systems*, 15(1):73–83, 2014.

[33] Robert Tibshirani. Regression shrinkage and selection via the lasso. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 267–288, 1996.

[34] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. 1991.

[35] Andreas Wedel, Daniel Cremers, Thomas Pock, and Horst Bischof. Structure-and motion-adaptive regularization for high accuracy optic flow. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 1663–1668. IEEE, 2009.

[36] Philippe Weinzaepfel, Jerome Revaud, Zaid Harchaoui, and Cordelia Schmid. Deepflow: Large displacement optical flow with deep matching. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1385–1392, 2013.

[37] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

[38] Jiadi Yu, Hongzi Zhu, Haofu Han, Yingying Jennifer Chen, Jie Yang, Yanmin Zhu, Zhongyang Chen, Guangtao Xue, and Minglu Li. Senspeed: Sensing driving conditions to estimate vehicle speed in urban environments. *IEEE Transactions on Mobile Computing*, 15(1):202–216, 2016.