Article

The International Journal of Robotics Research
1–21

DOI: 10.1177/0278364917710543
journals.sagepub.com/home/ijr

Deep tracking in the wild: End-to-end tracking using recurrent neural networks

Julie Dequaire, Peter Ondrúška, Dushyant Rao, Dominic Wang and Ingmar Posner

Abstract

This paper presents a novel approach for tracking static and dynamic objects for an autonomous vehicle operating in complex urban environments. Whereas traditional approaches for tracking often feature numerous hand-engineered stages, this method is learned end-to-end and can directly predict a fully unoccluded occupancy grid from raw laser input. We employ a recurrent neural network to capture the state and evolution of the environment, and train the model in an entirely unsupervised manner. In doing so, our use case compares to model-free, multi-object tracking although we do not explicitly perform the underlying data-association process. Further, we demonstrate that the underlying representation learned for the tracking task can be leveraged via inductive transfer to train an object detector in a data efficient manner. We motivate a number of architectural features and show the positive contribution of dilated convolutions, dynamic and static memory units to the task of tracking and classifying complex dynamic scenes through full occlusion. Our experimental results illustrate the ability of the model to track cars, buses, pedestrians, and cyclists from both moving and stationary platforms. Further, we compare and contrast the approach with a more traditional model-free multi-object tracking pipeline, demonstrating that it can more accurately predict future states of objects from current inputs.

Keywords

Deep Learning, tracking, recurrent neural networks, end-to-end, inductive transfer

1. Introduction

For an autonomous vehicle to operate safely and effectively, it must interpret its surroundings and predict the state of its environment over time. Such capabilities are relied upon by decision-making processes such as motion planning and control. In particular, the vehicle must be able to successfully track the state of objects in the environment through occlusion and predict how they will evolve in the future.

Commonly, tracking systems achieve this by employing multiple hand-engineered stages, such as object detection, semantic classification, data association, state estimation, motion modeling, and occupancy grid generation (Petrovskaya and Thrun, 2009; Vu et al., 2007; Wang et al., 2015). However, as robots begin to operate in increasingly complex environments this approach becomes more and more infeasible.

Recent research in machine learning has shown the ability of deep neural networks to capture complex structure, achieving state-of-the-art performance in numerous computer vision and natural language processing tasks (see, for example, Dahl et al., 2012; Krizhevsky et al., 2012; Wang et al., 2012). Such models usually require large, task-specific corpora of annotated ground-truth labels to master the desired task. However, such a data-intensive supervised learning paradigm is infeasible for object tracking in crowded urban environments, as we are required to learn a model of the environment without access to corresponding ground truth.

In this paper, we take an alternative approach and propose an end-to-end trainable framework to learn a model of the world dynamics in an entirely *unsupervised* manner. Building on the *deep tracking* paradigm introduced by Ondrúška and Posner (2016), we focus on the specific task of learning to track (and later classify) objects in complex and only partially-observable, real-world scenarios from both a *static* sensor as well as a *moving* platform.

For the case of a static sensor, we motivate and compare various neural network architectures and demonstrate improved tracking accuracy as compared to a more traditional model-free tracking approach. For a sensor exhibiting ego-motion, we extend the network architecture to account

Oxford Robotics Institute, University of Oxford, UK

Corresponding author:
Julie Dequaire, Oxford Robotics Institute, 23 Banbury Road, Oxford, OX2 6NN, UK.
Email: julie@robots.ox.ac.uk

for the movement of the sensor frame and demonstrate improved tracking accuracy as compared to prior art.

Our experimental evaluation leverages point cloud data gathered over 27 km of driving in busy urban environments of mixed static and dynamic occupancy, including cars, buses, pedestrians and cyclists. We further demonstrate that deep tracking learns rich latent representations of the environment that can be leveraged for other tasks such as semantic classification.

The exposition in this paper unifies and augments that of a number of conference and workshop publications and provides a more detailed description of the models used, with an extended discussion of results as well as a more elaborate experimental evaluation.

Section 2 highlights related work and Section 3 summarizes the problem definition. Section 4 provides a brief overview of the *deep tracking* framework as introduced by Ondrúška and Posner (2016). Section 5 describes the extensions required to apply deep tracking in real-world scenarios considering sensors on both static and dynamic platforms (Dequaire et al., 2016). Section 6 discusses how the *deep tracking* framework can be utilized for semantic classification by inductive transfer as first described by Ondrúška et al. (2016). Section 7 presents an empirical evaluation of our method. We conclude in Section 8.

2. Related work

This paper addresses the problem of tracking objects in the workspace of a mobile robot. Traditional approaches to this problem comprise a multi-stage pipeline, with separate components to perform object detection, associate new measurements with existing tracked objects, and perform state estimation for each individually tracked object. Several such approaches perform *model-free* tracking (Vu et al., 2007; Wang et al., 2015; Yang and Wang, 2011), and assume a general motion model for objects, but suffer in terms of robustness. In contrast, *model-based* techniques (Arras et al., 2007; Petrovskaya and Thrun, 2009; Zhao and Thorpe, 1998) explicitly consider the type of object being tracked, which limits their generality but improves tracking performance for the object classes considered. In all of these cases, however, the use of multiple hand-engineered stages in the framework is cumbersome, requires manual tuning of parameters, and introduces unnecessary additional failure modes to the tracking process.

Ondrúška and Posner (2016) propose to replace these multiple stages with a single end-to-end learning framework known as *deep tracking*, by leveraging neural networks to directly learn the mapping from raw laser input to an unoccluded occupancy grid. Here, the end-to-end model implicitly performs the individual steps implemented in the traditional multi-stage pipeline, without the need for hand engineering or tuning of parameters for each component.

While deep tracking leverages a recurrent neural network (RNN, Medsker and Jain, 2001) to capture the state and evolution of the world in a sequence of laser frames, Choi et al. (2016) follow a different approach using *recurrent flow networks*. The model explicitly encodes a range of velocities in the hidden layers of a RNN, and uses Bayesian optimization to learn the network parameters responsible for updating the velocity estimation and occupancy prediction. However, the model is not trained to explicitly track objects through occlusion. Another work (Byravan and Fox, 2016) utilizes deep networks to capture rigid body motion between objects in pairs of 3D point clouds.

Deep tracking is related to deep learning approaches applied to predictive video modeling (Lotter et al., 2016; Patraucean et al., 2015), in that it is trained to predict the future state of the world based on current input data. This is of vital importance to achieve accurate prediction and tracking, as to successfully predict the future location of dynamic objects in the scene, the model must implicitly store the position and velocity of each object in its internal memory state.

The original work on deep tracking (Ondrúška and Posner, 2016) eradicates the need to design individual components by hand, and demonstrates the efficacy of the framework on a synthetic dataset comprised of circular objects exhibiting linear motion in a cluttered environment. Ondrúška et al. (2016) then extend this to real-world data from a static vehicle, which introduces additional complexity due to the different sizes, shapes, and velocities of objects in the environment, and larger occlusions in the raw sensor input. To achieve this, the RNN-based models proposed by Ondrúška and Posner (2016) were scaled up for real-world application, and alternative architectures proposed.

As both of these approaches assume a static sensor, the framework is then extended to operate on a moving platform by Dequaire et al. (2016). This is a challenging task because it introduces an array of complex relative motions between the vehicle and objects in its environment. If vehicle motion is ignored in this scenario, the model is forced to learn all possible motion interactions between the vehicle and its environment as if the vehicle were stationary. For the moving platform, estimates of visual ego-motion are used as a proxy for vehicle motion, and incorporated into the deep tracking framework by exploiting *spatial transformer* modules (Jaderberg et al., 2015), which allow the internal belief state of the RNN to be transformed according to the estimated ego-motion.

It is also possible to exploit the fact that the learned representation captures latent higher-order information in the data, such as the location, shape, and velocity of an object, which is also pertinent to semantic classification tasks. This was initially explored by (Ondrúška et al., 2016), and means that we can infer semantic labels for the tracked objects using only a small amount of labeled data. This is a form of *inductive transfer* of knowledge between machine learning tasks (Pan and Yang, 2010). In the context of neural networks it has been successfully applied to a range of tasks,

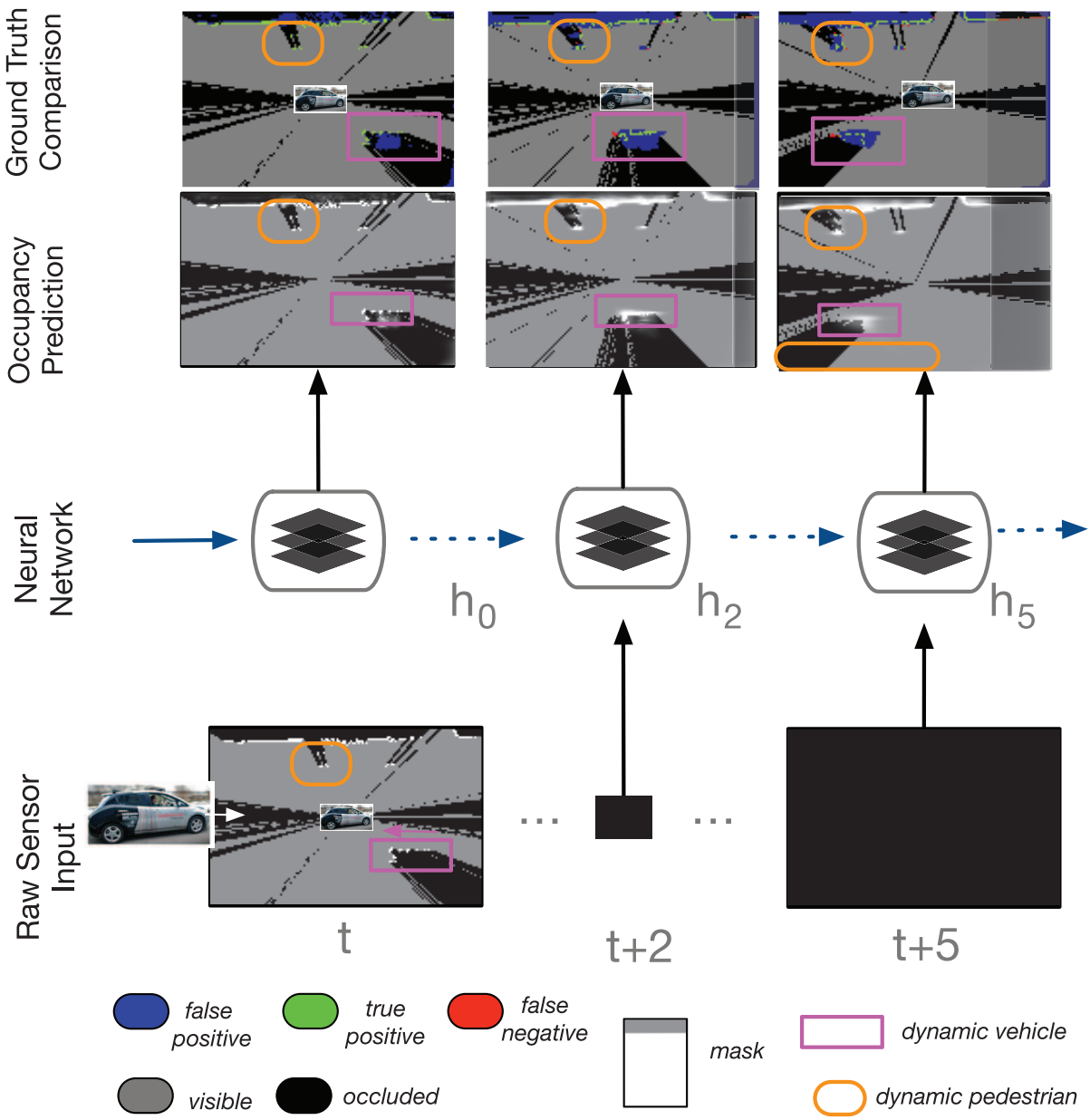

Fig. 1. A typical training sequence where input frames from $t+1$ to $t+5$ are blacked out to simulate total occlusion. The unoccluded occupancy map is predicted directly from the input grid data, allowing objects to be tracked through occlusion and in future frames. Any observed false positives, which represent the network's capacity to predict in occlusion, are therefore beneficial. Comparison to the visible ground truth shows that the model is able to capture the dynamics of the moving vehicle (pink rectangle) and moving pedestrians further occluding one another (orange contours). A quantitative evaluation is conducted in Section 5.2.

in the areas of multi-task learning (Caruana, 1995; Mitchell and Thrun, 1993) and in the form of unsupervised pre-training and supervised fine-tuning (Le, 2013; Pennington et al., 2014).

3. Problem formulation

This work addresses the challenge of uncovering the true, *unoccluded* state of the world, in terms of a 2D occupancy grid $\mathbf{y}_t$, given a sequence of *partially observed* states of the environment $\mathbf{x}_{1:t}$ computed directly from raw LIDAR measurements. The input observation at each time step t, $\mathbf{x}_t \in \{0,1\}^{2\times M\times M}$, is represented as a pair of discretized 2D binary grids of size $M \times M$, parallel to the ground and centered on the sensor frame. The first of these matrices encodes *visibility*: whether a cell is directly observable by the sensor at time t; while the second matrix encodes *occupancy*: whether a cell is observed to be free (value of 0) or occupied (value of 1). These two matrices are denoted as $\mathbf{x}_{t,vis}$ and $\mathbf{x}_{t,occ}$ respectively. The true unoccluded map is

another grid $\mathbf{y}_t \in \{0,1\}^{M\times M}$ encoding the occlusion-free state of the world.

Thus, the goal is to solve for $P(\mathbf{y}_t|\mathbf{x}_{1:t})$, the conditional probability of the complete, unoccluded state of the world at time t given a sequence of partial observations at all previous time steps. Under the same formulation, future states can also be predicted by solving for $P(\mathbf{y}_{t+n}|\mathbf{x}_{1:t})$ after masking the future inputs, i.e. $\mathbf{x}_{t+1:t+n} = \mathbf{0}$.

To estimate $P(\mathbf{y}_t|\mathbf{x}_{1:t})$, we use the deep tracking framework (Ondrúška and Posner, 2016). However, in its original formulation, this framework is only demonstrated on simulated scenarios composed of simple geometric objects using a simple network architecture. A number of modifications to this architecture are needed to scale up its capacity to deal with complex and dynamic real-world data, as well as scenarios where the sensor may additionally be mounted on a moving platform, as commonly encountered in robotics applications.

The tracking problem of solving for $P(\mathbf{y}_t|\mathbf{x}_{1:t})$ can also be further extended to predicting the scene semantics $P(\mathbf{c}_t|\mathbf{x}_{1:t})$. This takes the form of predicting one of the K objects for each cell. The semantic map $\mathbf{c} \in \{1,\ldots,K\}^{M\times M}$ can reflect object classes such as *pedestrian, cyclist, car, static background*, or whether an occupied cell belongs to a *dynamic* or *static* object.

Next, we briefly outline deep tracking as a solution to the first part of the problem; namely to estimate $P(\mathbf{y}_t|\mathbf{x}_{1:t})$.

4. A brief review of deep tracking

Deep tracking (DT) is a framework initially introduced by Ondrúška and Posner (2016) to model the conditional distribution $P(\mathbf{y}_t|\mathbf{x}_{1:t})$ using a RNN (Medsker and Jain, 2001). Motivated by recursive state estimation, the underlying process modeled by the deep tracking framework is assumed to exhibit the Markov property. That is, the latent state at time t, denoted by $\mathbf{h}_t$, captures the complete information required to predict the output $\mathbf{y}_t$, which may denote scene appearance and dynamics, locations of all objects, their shapes, and velocities. Thus, we have

$$P(\mathbf{y}_t|\mathbf{x}_{1:t}) = P(\mathbf{y}_t|\mathbf{h}_t) \tag{1}$$

Evolution of this latent state, including the propagation of model dynamics and integration of new sensor measurements, is modeled by an update operation

$$\mathbf{h}_t = f(\mathbf{h}_{t-1}, \mathbf{x}_t) \tag{2}$$

Crucially, both the latent state update $f(\mathbf{h}_{t-1}, \mathbf{x}_t)$ and the output decoding step $P(\mathbf{y}_t|\mathbf{h}_t)$ are modeled and trained jointly as parts of a single neural network. The prediction and update steps in equations (1) and (2) can then be performed repeatedly as part of a recurrent neural network that updates the current belief, or *network memory* state, $\mathbf{h}_t$, and uses it to predict the unoccluded output $\mathbf{y}_t$. In this way, the model can be used for online filtering of the sensor input.

Since the true state of occluded objects is not known in real-world scenarios, the output ground-truth $\mathbf{y}_t$ is not readily available. To circumvent this issue, we train the network in a self-supervised fashion. To do this, consider that predicting the movement of objects under occlusion at time t is similar to predicting a *future* state $\mathbf{y}_{t+n}$ without any future input provided to the network, i.e. $\mathbf{x}_{t+n} = \mathbf{0}$. The lack of input observations equates to complete occlusion of the scene. Given that the *observable* ground truth is available, we alter the problem of outputting $\mathbf{y}_{t+n}$ to that of predicting the parts of $\mathbf{y}_{t+n}$ that are directly observable, denoted $\mathbf{y}'_{t+n}$, which is equivalent to the observed input $\mathbf{x}_{t+n}$. Training the network to instead predict the probability of the observable ground truth $P(\mathbf{y}'_{t+n}|\mathbf{x}_{1:t})$ is equivalent to only back-propagating errors in the observable parts of the scene. In other words, the network is trained to correctly predict the subset of the ground-truth occupancy present in the *future input*. The training procedure is illustrated in Figure 2. As demonstrated by Ondrúška and Posner (2016), an important consequence of this training strategy is that, at deployment, the trained network starts to correctly imagine objects and their movement in the occluded regions (Figure 15). This is because the situation with an occluded input at deployment is similar to that at training when no input was provided at all for the future time $t + n$, and the network was trained to predict the observable regions.

5. Deep tracking in the wild

The simple recurrent neural network proposed by Ondrúška and Posner (2016) was demonstrated to be sufficient for the simulated dynamic scenario evaluated in that work. However, an effective deployment in real-world robotics applications poses a set of challenges for which a more appropriate architecture must be chosen. In the following sections, we present our choice of extensions to scale up the baseline framework's capacity to deal with a complex real-world robotics task, for sensors mounted on either stationary or moving platforms. We also extend our solution to estimate the scene semantics $\mathbf{c}_t$ via the application of the principle of inductive transfer.

5.1. Deep tracking from a static sensor

For application to real-world scenarios, the framework must specifically be able to simultaneously track objects of different sizes such as vehicles and pedestrians, and effectively remember information for long periods of time to deal with occlusions. In the case of a static sensor, it can additionally be valuable to learn to exploit place-dependent information such as the presence of static obstacles.

An overview of the proposed network is shown in Figure 3. The input $\mathbf{x}_t$ at time t is processed by multiple layers. The activations for each layer l at time t are a function of the activations of layer $l-1$ below, as well as its own activations at time $t-1$, and thus implementing the

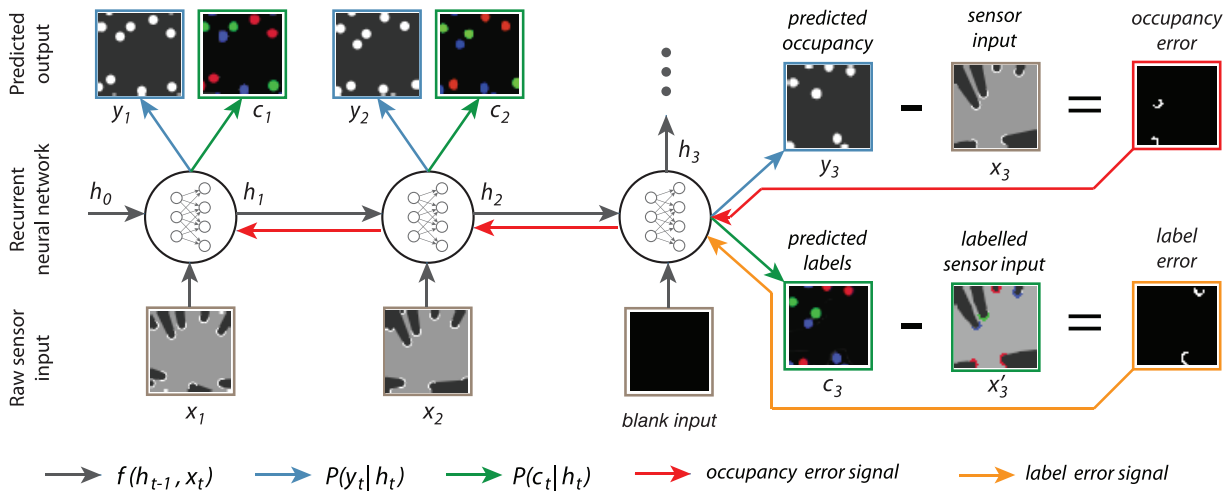

Fig. 2. Training of the recurrent neural network to produce both space occupancy $\mathbf{y}_t$ and semantic labels $\mathbf{c}_t$. The network is trained to predict outputs consistent with future inputs. This allows training without the need for ground-truth information of the full, unoccluded scene. First, the network learns how to track by predicting correct occupancy using large amounts of unlabeled data, then a small set of labeled data is used to induce semantic classification.

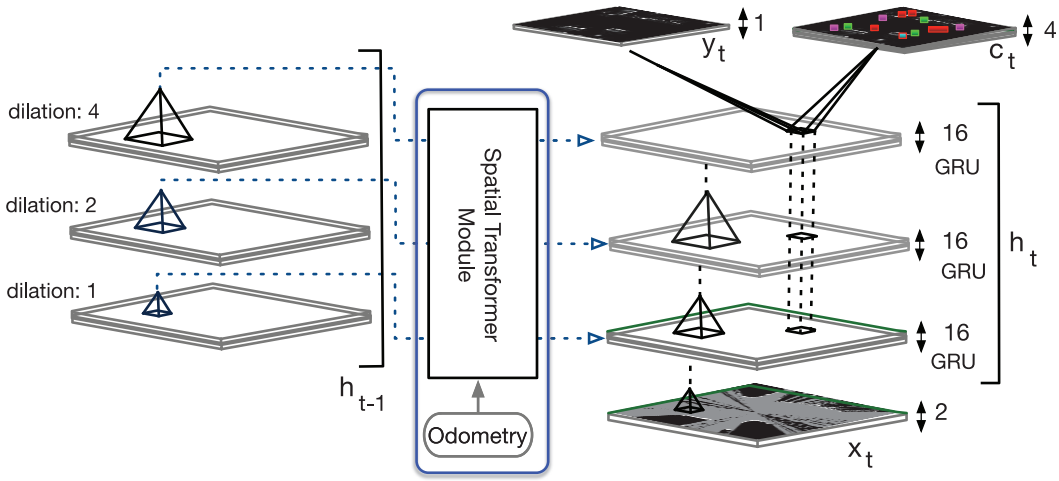

Fig. 3. The architecture proposed for tracking and semantic classification. It features dilated convolution, enhanced static and dynamic memory capabilities and produces information of both cell occupancy and its semantic class. The STM is only utilized in a moving vehicle scenario. These architectural choices are motivated in the main text.

recurrence. The incorporation of dilated convolutions (Yu and Koltun, 2015) allows the model to simultaneously track objects of very different sizes, such as buses, cars, and pedestrians. A variant of gated recurrent units (GRUs) (Cho et al., 2014; Xingjian et al., 2015) allows the network to extract and remember information from the past and use it to predict occluded objects or future states of the world. Finally, an additional static memory can also be utilized, in which the network is given the capacity to learn individual pixel-wise biases that are added to the output of each convolution. A simple convolutional sigmoid function decoder is then applied to the feature maps from all the layers of the network, to obtain the final cell occupancy output grid $\mathbf{y}_t$. For model performance comparison, we also experiment with architectures that decode only the 16 feature maps of the last hidden unit. We specify this in the text when applicable. These features are described in greater detail in the following sections.

5.1.1. Multi-scale convolution. In order for the network to correctly predict the occupancy and label of a moving object at location i, the object must fall within the *receptive field* of the hidden unit in the final layer. The *receptive field* specifies the part of the input data that can affect the activation value of the hidden unit. For a traditional convolution, the receptive field is the $K \times K$ neighborhood where K is the size of the convolution kernel. The size of the receptive field, however, limits the size (in terms of the input data) of objects that can be effectively tracked. This is unrealistic in real-world settings, as dynamic objects can be of vastly different sizes, from pedestrians to buses. This can be

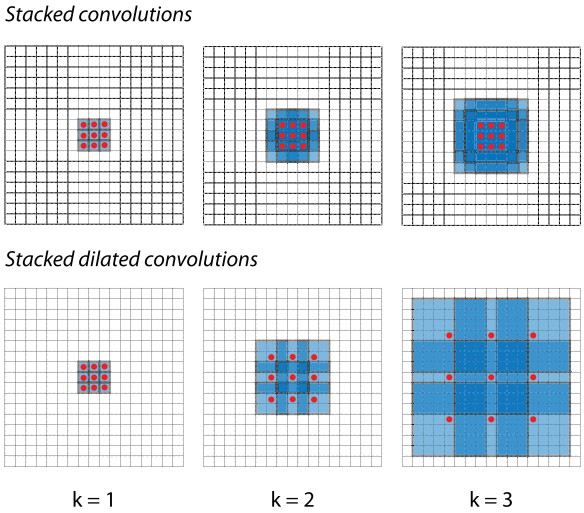

Fig. 4. Multi-scale context aggregation preserve the image resolution by stacking *dilated convolutions* (Yu and Koltun, 2015). At layer k the red pixels are convolved with a skip of $2^{k-1}-1$ pixels. This results in exponential growth of the blue *receptive field* **(bottom)** as opposed to stacking classical convolutions, which result only in linear growth **(top)**.

ameliorated by increasing the receptive field size, either by increasing the kernel size or stacking multiple convolutions on top of one another. However, this creates a computational challenge as the number of parameters and computational complexity grows quadratically with K in the first case and linearly in the second case.

We circumvent this issue by instead exploiting dilated convolutions (Yu and Koltun, 2015) through the network layers, where the receptive field grows exponentially with the number of layers. That is, we perform the classical 3×3 convolution but skip $2^{k-1}-1$ pixels in between convolved pixels at each layer k, as illustrated in Figure 4. This results in a $(2^{K+1}-1) \times (2^{K+1}-1)$ receptive field at final layer K. This dilated convolution is then used as a building block to implement the *dynamic memory* described in the following section. This technique affords the ability to capture structure at multiple scales while reducing the amount of weights necessary to obtain the desired receptive field.

5.1.2. Dynamic memory. To be able to track a moving object through extended periods of occlusion, the network must keep in memory the location of the object and other properties such as its shape and velocity. Prior art on recurrent neural networks stresses the importance of specially dedicated units such as long short term memory (LSTM) to support information-caching (Hochreither and Schmidhuber, 1997), to ensure training is not hampered by the vanishing gradient problem (Pascanu et al., 2012). Inspired by Xingjian et al. (2015), we implement a convolutional variant of *GRUs* (Cho et al., 2014) as the processing step at each layer. GRUs are computationally simpler than LSTM units, and have been shown to exhibit a very similar performance. The output of each unit is given by the weighted combination of its previous output at time $t-1$ and a candidate memory $\bar{\mathbf{h}}_t$ computed from the output of the layer below. The amount of information retained or forgotten can be modulated by the unit using the reset and forget gates $\mathbf{r}_t, \mathbf{f}_t$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{xz} * \mathbf{x}_t + \mathbf{W}_{hz} * \mathbf{h}_{t-1} + \mathbf{b}_z) \quad (3)$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{xr} * \mathbf{x}_t + \mathbf{W}_{hr} * \mathbf{h}_{t-1} + \mathbf{b}_r) \quad (4)$$

$$\bar{\mathbf{h}}_t = \tanh(\mathbf{W}_{xh} * \mathbf{x}_t + \mathbf{r}_t \circ \mathbf{W}_{hh} * \mathbf{h}_{t-1} + \mathbf{b}_h) \quad (5)$$

$$\mathbf{h}_t = \mathbf{f}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{f}_t) \circ \bar{\mathbf{h}}_t \quad (6)$$

Here $*$ denotes dilated convolution described earlier and $\circ$ denotes element-wise multiplication.

5.1.3. Static memory. Another extension of the model is to allow each cell to learn a unique and universally accessible piece of information different from all other cells. This is achieved by biases $\mathbf{b}_z$, $\mathbf{b}_r$, and $\mathbf{b}_h$ in equations (3) to (5) which are not a per-layer constant as in the case of classical convolution, but are learned individually for each neuron during training. As shown in Section 7 this allows the network to learn place-specific information such as the static occupancy of the cell or the usual motion patterns and classes in a particular area, which can then be used to aid the network prediction.

5.2. Deep tracking from a moving vehicle

In order to track a dynamic scene from a moving platform, it is necessary to address the additional challenge of decoupling the motion of the vehicle from the motion of dynamic objects in the environment. This ensures that the internal memory state of the RNN only captures the state of the environment, without superimposing the motion of the vehicle itself.

For a static vehicle, information related to an object located at index $\{i,j\}$ in the input $\mathbf{x}_t$ is contained within a spatial neighborhood $\{i+\Delta i, j+\Delta j\}$ in the layers of the latent representation, with the neighborhood $\Delta_{i,j}$ depending on the receptive field of each hidden unit. The latent state update would then pass the information along spatially in accordance with the observed relative motion of the input object (Figure 5). In the static scenario, the dynamics of the scene as viewed from the world frame are coherent with that viewed from the local sensor frame.

In the dynamic vehicle scenario, however, the latent representation would additionally need to account for the sensor's ego-motion, as it affects the position of the object relative to the sensor frame. Although this is a major drawback of the baseline *deep tracking* architecture, it can be compensated for by transforming the memory state in accordance to the estimated ego-motion. That is, a static obstacle at position $\{i_{t-1}, j_{t-1}\}$ in the sensor frame at $t-1$, will be

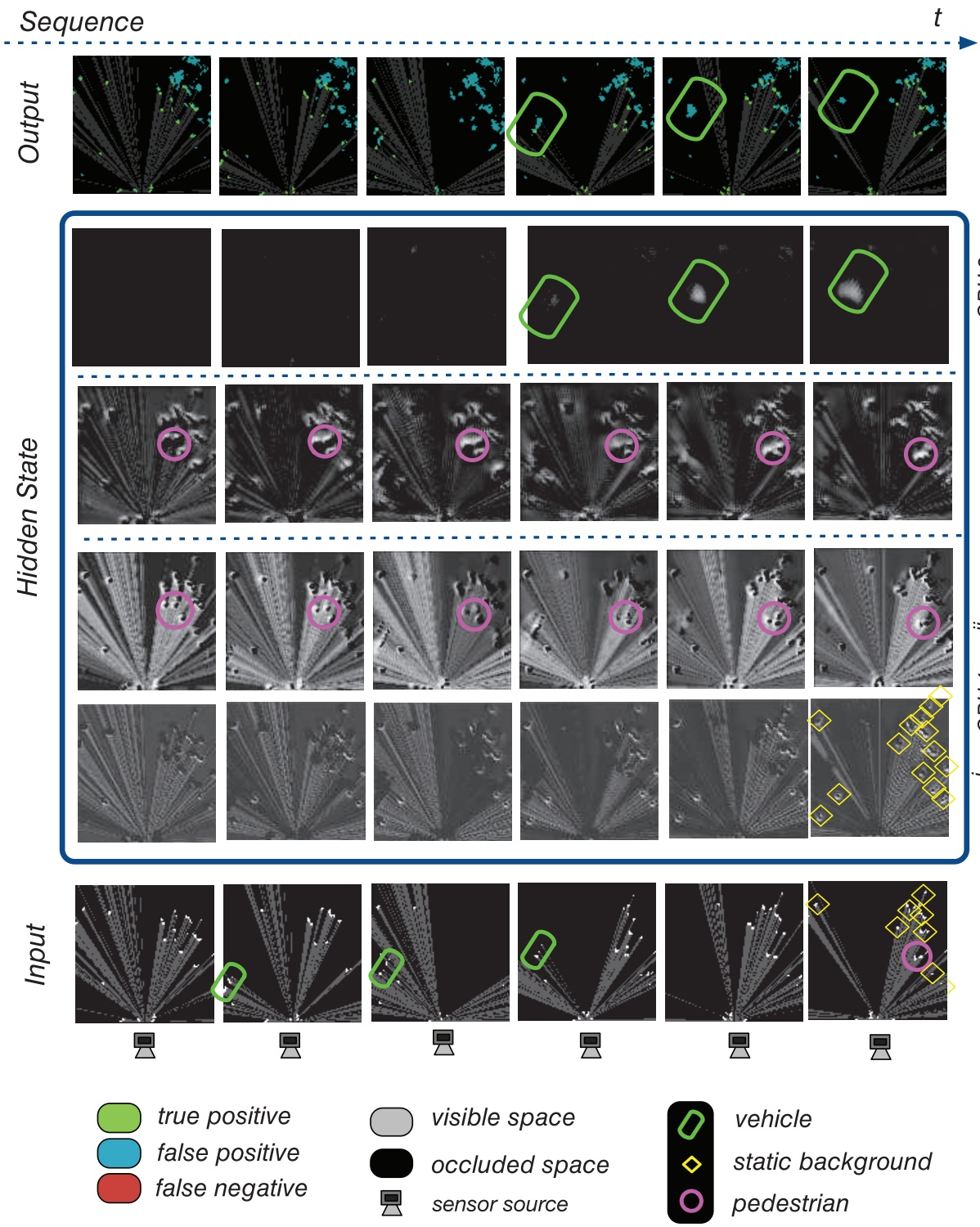

Fig. 5. Example of outputs produced by the system (GRU3DilConv_48) along with a selection of activations in the hidden layers. The figure is best read from bottom to top, with the bottom layer providing the raw sequence input to the network along with ground truth annotation. As highlighted in color-coded circles, the network is able to propagate the assumed motion of the objects even when in complete occlusion. The sample hidden layer activations shown highlight the fact that lower layers in the hidden units (corresponding to a low dilation of the convolutions) capture the motion of small and slow moving objects such as pedestrians (e.g pink circle) and static background (e.g yellow squares), whereas a higher-level layer learns to detect moving vehicles (green contour).

transformed to position $\{i_t, j_t\}$ in the sensor frame at time t according to

$$[x_t, y_t, 1]^T = T_{t,t-1} \times [x_{t-1}, y_{t-1}, 1]^T \quad (7)$$

where $T_{t,t-1}$ is the $\mathbb{SE}(2)$ forward transformation of the sensor source frame at $t-1$ into the sensor destination frame at t. This formulation naturally extends to 3D motion with $\mathbb{SE}(3)$. $\mathbb{SE}(2)$ and $\mathbb{SE}(3)$ refer to the Lie groups representing rigid transformations in respectively 2D and 3D space.

The network can be encouraged to decouple ego-motion and object motion in this way by introducing a *spatial transformer* (Jaderberg et al., 2015) module (STM) in the hidden state (Figure 3). The original STM introduced by Jaderberg et al. (2015) is a learnable module that actively transforms feature maps to aid tasks such as the classification of spatially distorted datasets. However, in the context of tracking from a moving platform where ego-motion is readily available (e.g. from visual odometry algorithms), the STM can simply utilize the forward transformation $T_{t,t-1}$ obtained from ego-motion estimates, transforming the hidden feature maps centered in the sensor frame at time $t-1$ into the sensor frame at time t (equation (7)).

As the visual odometry module provides $\mathbb{SE}(3)$ transformations for the ego-motion of the vehicle, we convert these to $\mathbb{SE}(2)$ transforms for use in the STM. This is achieved by selecting the x, y and yaw planar components of the vehicle frame, where the z-axis is locally vertical to the ground. As this is performed at every timestamp, we find that the approximation involved introduces negligible error in the sequences considered. We further discuss the choice and limitations involved with this implementation in Section 8.

The STM is thus incorporated within the hidden state, transforming all feature maps $\mathbf{h}_{t-1}$ at time $t-1$, into the sensor frame at time t. This is performed before the update of the memory with the new incoming input $\mathbf{x}_t$. We illustrate this in Figure 3.

6. Semantic classification through inductive transfer

In this section, we extend our solution to the partial problem $P(\mathbf{y}_t|\mathbf{x}_{1:t})$ presented in Section 4, to the full problem of simultaneously estimating both occlusion-free occupancy and scene semantics $\mathbf{y}_t, \mathbf{c}_t$. We show that this can be achieved relatively easily by exploiting the knowledge the network has already learned to predict $\mathbf{y}_t$, through the principle of inductive transfer (Pan and Yang, 2010). The significance of this is that *only a small amount* of labeled training data is needed to allow the same network to master this additional task.

The clue resides in the hidden representation $\mathbf{h}_t$ learned in the unsupervised training for tracking, which can be viewed as a universal descriptor of the state of the world. It captures not only the positions of individual objects, but also their motion patterns, shapes and other properties necessary for the successful prediction of scene dynamics.

Fig. 6. Location of the experiment from the robot's point of view with a superimposed illustration of laser measurements. The area is occupied by a variety of different dynamic objects such as pedestrians, cyclists, and cars.

Because the network was trained to perform well in this task, a reasonable assumption to make is that any information necessary for the prediction of the position of the objects in the near future must be already contained in this hidden representation. The semantic class of an object falls in this category as different objects differ mainly in their shape and motion patterns. Similar to predicting $\mathbf{y}_t$ from $\mathbf{h}_t$ in equation (1), extracting $\mathbf{c}_t$ can be achieved simply by building a classifier to predict $P(\mathbf{c}_t|\mathbf{h}_t)$. This is achieved by employing a simple convolutional softmax function decoder to decode the hidden state $\mathbf{h}_t$ to a semantic grid $\mathbf{c}_t$ of 1 of K class labels.

7. Results

7.1. Deep tracking from a static sensor

In this section we demonstrate the efficacy of the proposed system in the task of tracking in complex, real-world scenarios. We show that the trained network is able to track a variety of different objects even through complete occlusion, and is able to predict the evolution of the scene in the near future. The achieved tracking accuracy is superior to the original architecture presented by Ondrúška and Posner (2016), as well as to that of an alternative state-of-the-art model-free tracking solution targeted at the same problem (Wang et al., 2015).

7.1.1. Dataset. The dataset for this analysis was obtained with a robotic platform equipped with a *Hokuyo UTM-30LX* 2D laser scanner, and consists of 75 min of data acquired while stationary in the middle of a busy urban intersection, as depicted in Figure 6. This was subsampled at 8 Hz and split into a 65 min unsupervised training set and a 10 min long test set to measure the occupancy prediction performance. The dataset is challenging due to the dense traffic of

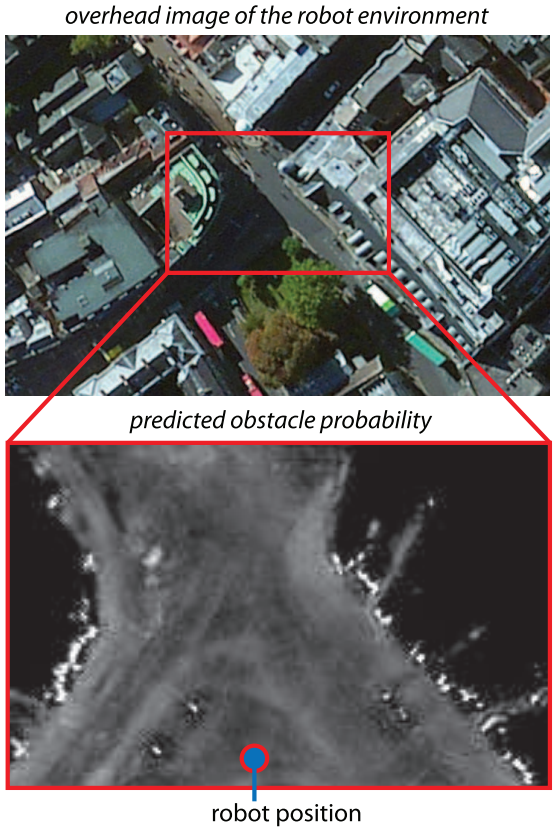

Fig. 7. Trained network output when provided no input **(bottom)** and corresponding aerial view of the robot workspace **(top)**. The ability of the network to learn per-pixel information allows adaptation to the training environment. This allows the network to confidently predict the position of static obstacles such as buildings, as well as the probability of any given cell being occupied even without any sensor input. Pavements show higher probabilities than the center of the roads. For clarity of visualization, we show here the log of the probabilities of occupation.

buses, cars, cyclists, and pedestrians, which results in extensive amounts of occlusion. Consequently, at no point in time is the complete scene fully observable.

7.1.2. Network parameters. The occluded occupancy map used as input to the network $\mathbf{x}_t$ is computed from raw 2D laser scans by performing ray-tracing. Cells corresponding to a laser range measurement are marked as occupied, all cells from the sensor origin to the ray ending at the occupied cell are marked as free, and cells beyond the ray are marked to be unobserved. Thus, a 100×100 grid is constructed, with the origin centered at the sensor location. Each cell covers a 20×20 cm^2 area, and the input grid thus spanning a total area of 20×20 m^2. We consider an architecture with three hidden layers of 16 GRU feature maps each. Computation of the hidden state consists of 3×3 convolutional filters, applied on the three layers (from bottom up) with strides of one, two, and four pixels, and receptive fields of 3×3, 7×7, and 15×15, respectively. With a hidden state consisting of 48 feature maps, the additional static memory contributes to $H \times W \times D = 489,648$ of the total $1,506,865$ parameters of the network.

7.1.3. Network training. The network is presented the input sequence $\mathbf{x}_{1:t}$ and trained to predict the next n input frames $\mathbf{x}_{t+1:t+n}$. The binary cross-entropy loss is calculated and backpropagated only on the ground truth available, i.e. the *visible* part of the space. This is achieved by simply masking the output prediction $\mathbf{y}_t$ with the visible input $\mathbf{x}_{t,vis}$ and multiplying the resulting grid element-wise with the occupancy part grid $\mathbf{x}_{t,occ}$. Since the loss encourages the model to predict future states of the environment, the model is forced to capture the motion of each object in the hidden representation.

The training set was split into mini-batch sequences of 40 frames (5 s). For every mini-batch, the network was shown 10 frames and trained to predict the next 10 frames, leading to two such sequences per 40-frame mini-batch. This length of sequence covers the typical lengths of occlusions observed but could be tuned accordingly on other datasets.

Our model architectures are implemented in the Lua programming language using the Torch library.[1] Models are trained on an NVIDIA Tesla K40 graphics processing unit (GPU) until convergence, using the unsupervised training procedure described in Section 3. The Adagrad optimizer is used for a stochastic gradient descent with an initial learning rate of 0.01, and the loss is monitored on the validation set to perform early stopping and prevent over-fitting.

7.1.4. Benchmarking against an existing approach. To justify the proposed end-to-end system, its performance is assessed in comparison to a more traditional multi-stage pipeline. In particular, we evaluate the ability of the model to predict the future location of dynamic objects and compare it with a recently proposed state-of-the-art approach (Wang et al., 2015) based on model-free tracking of dynamic objects using a Kalman filter. This method also operates on raw laser scans, but explicitly performs clustering, data association, and velocity estimation of moving objects. This information is then used to predict the positions of individual points in the future. The parameters of this method are tuned on the training set, and the outputs on the test set are converted into occupancy grids for comparison with the proposed method.

7.1.5. Quantitative results. In the first experiment, we look to quantify the gain in performance achieved by scaling up the original *deep tracking* network with the proposed architecture, and individually explore the benefits of each of the improvements proposed. Accordingly, we compare the performance of a number of different architectures, ranging from the original (Ondrúška and Posner, 2016) to the proposed model, on the task of predicting the observable near-future scene occupancy $\mathbf{y}'_{t+1:t+n}$, given the input sequence

$\mathbf{x}_{1:t}$. The predicted output occupancy $P(\mathbf{y}_{t+n}|\mathbf{x}_{1:t})$ is compared to $\mathbf{x}_{t+n,vis} = \mathbf{y}'_{t+n}$, which corresponds to the visible (and therefore known) part of the world at time $t + n$. A threshold of 0.5 is used to determine whether a cell is predicted as occupied or free, and an F1 measure computed for each frame.

Figure 8 reports the F1 measures computed on each of $n = 10$ blacked out future frames, given the 10 frames in the past, for numerous model architectures. Intuitively, the predictions of all models are poorer over time, as the uncertainty of the state of the world increases with the prediction horizon. There is a considerable increase in performance when utilizing deep tracking architectures compared to the model-free tracking pipeline from the work of Wang et al. (2015), illustrating the efficacy of the proposed end-to-end approach.

Figure 8(a) investigates the effect of scaling up the original deep tracking framework by Ondrúška and Posner (2016) in terms of the model capacity and memory unit (GRU vs RNN). We find that merely increasing the capacity of the original RNN architecture (RNN16_encoder) from 16 to 48 feature maps (RNN48_encoder) does not yield any significant performance increase, indicating that the capacity of the network is not the limiting factor. Removing the eight-feature map encoder, however, increases performance (RNN48) suggesting there is no added value in encoding the input. We discard the encoder in all subsequent models. We investigate the effect of a multi-stage processing of the input by separating the 48 feature maps of the hidden state into three hidden layers of 16 feature maps each (RNN48_split), similarly to the architecture proposed in Figure 3. This model increases the output receptive field from 5×5 to 13×13 and yields poorer performance. However, we find that changing the internal memory state from RNN to *GRUs* leads to the most significant performance increase (GRU3_16_ker5). Similarly to RNN48_split, GRU3_16_ker5 decodes only the last 16 feature maps of the last hidden layer to the output occupancy $\mathbf{y}_t$, and considers kernels of size 5×5. We consider GRU architectures in all subsequent models.

Figure 8(b) investigates the effect of varying output receptive fields and utilizing dilated convolutions. Similar to what was observed in the RNN architecture, we do not find any significant model performance increase between an output receptive field of 7×7 (GRU3_16_ker3 with 3×3 kernels) and output receptive field of 13×13 (GRU3_16_ker5 with 5×5 kernels). Building upon GRU3_16_ker5, replacing the standard convolutions with dilated convolutions (GRU3DilConv_16) achieves a similar performance for a comparable receptive field of 15×15 using 3×3 kernels. GRU3DilConv_16 achieves commensurate if not better performance (with fewer parameters) than an architecture achieving similar receptive fields but with dense kernels (GRU3_16_A). However, further increase of the receptive field to 19×19 (GRU3_16_ker7 with kernels of 7×7) results in a poorer performance than that of GRU3_16_ker5. A small performance increase is obtained by reducing the number of parameters with dilated convolutions (GRU3DilConv_16_A). This model retains dense kernels of 7×7 in the feature maps of the first hidden layer, and uses dilation with kernels 3×3 in the rest of the hidden state. This maintains receptive fields of 7×7, 11×11, and 19×19 in the hidden units, which is comparable to those of GRU3_16_ker7. Performance nonetheless remains below that of its lower receptive field counterpart GRU3DilConv_16. These results suggest that a smaller output receptive field of 7×7 suffices in capturing the overall scene dynamics. We hypothesize that too large an output receptive field may contribute to a loss of resolution. Further, we find that dilated convolutions achieve similar receptive fields in a parameter-efficient manner, without loss of performance. Both dilated architectures considered here allow for a parameter factor reduction of nearly three.

Figure 8(c) suggests that performance of the full model obtained when adding static biases (GRU3Dil ConvBias_16) remains commensurate to that of GRU3 DilConv_16, and the learned static bias values may convey useful information such as that of the static background layout. This is for example valuable when learning the scene semantics as presented in Section 7.3.

One key departure from a traditional convolutional architecture would be to consider the full hidden state (all three layers with 16 feature maps each) when decoding to the output occupancy $\mathbf{y}_t$, rather than the feature maps in the final layer. We compare both of these cases for our best two models (GRU3DilConvBias_16 and GRU3DilConv_16), and find that these four architectures are commensurate in capturing the scene dynamics and bring favorable improvement to the original architecture. However, similarly to the role of static biases, we find that decoding the full hidden state favorably influences the model's capacity to classify the scene semantically. We discuss this further in Section 7.3.

7.1.6. Qualitative results. To measure the effectiveness of the desired ability to learn place-specific information, we visualize the network prediction $\mathbf{y}_t$ without providing it with any input as displayed in Figure 7. Even without input sensor information, the network is able to provide an estimate of the expected occupancy probabilities, which is higher at the locations of static obstacles and at crowded areas of the scene such as pavements. As no propagation of the information through the network occur this is clearly only made possible by the ability of the network to remember this information in its static memory during training. This visualization was performed on a higher resolution network after 10 epochs of training.

To better understand what the network has learned, we also perform qualitative analysis of a typical 3 s test sequence, visualizing the network output and selected hidden state feature maps in Figure 5. The network is able to track pedestrians through full occlusion and the unobserved

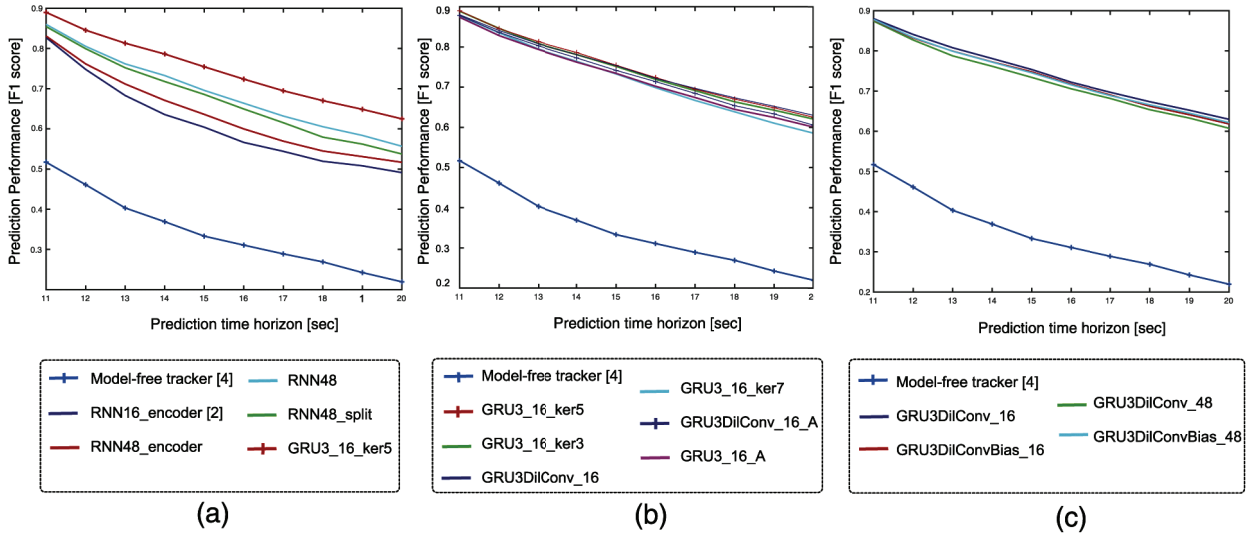

Fig. 8. F1 scores of network architectures when attempting to predict the future occupancy of the scene in a 1.25 s time horizon. The F1 measure is computed with a threshold of 0.5 when considering a cell to be predicted as occupied or free. (a) Investigating model capacity and memory unit suggests that incorporating *GRUs* accounts for the most significant performance gain. (b) Investigating output receptive field and dilated convolutions suggests that dilated convolutions achieve commensurate performance in a parameter-efficient manner and that small output receptive fields (7×7) suffice. (c) Investigating static biases and full hidden state decoding suggests that these do not affect model performance.

hallucinated tracks are represented in blue in the output sequence.

It can be seen that some of the first layer feature maps (GRU1) appear to have learned to capture the static background during the sequence with:

(a) a stationary set of activations;
(b) track moving pedestrians, as highlighted through the pink circles.

The second layer (GRU2) also captures the motion of pedestrians moving upwards to the left, while, interestingly, the GRU3 unit is activated only on the car that appears to the top right at frame two (indicated by the orange box). This provides empirical support for the use of dilated convolutions, which make it possible to capture patterns of varying width with limited computation.

In general, we observe that information regarding objects in the scene are captured in the hidden state, and move spatially through the feature maps according to the motion of the object in the input.

7.2. Deep tracking under ego-motion

This section evaluates our best tracking model (GRU3DilConv_48) which features GRU3 units and dilated convolutions (baseline DT), with an equivalent architecture which additionally incorporates the STM into the into the hidden belief state as represented in Figure 3. Both these models decode the whole of the hidden state to the output. This indicates the importance of the STM when deploying the architecture on a moving vehicle. As with the static case, we evaluate the ability to predict future frames given masked future input, and qualitatively discuss the occlusion-free tracking performance on a series of examples selected from the test set.

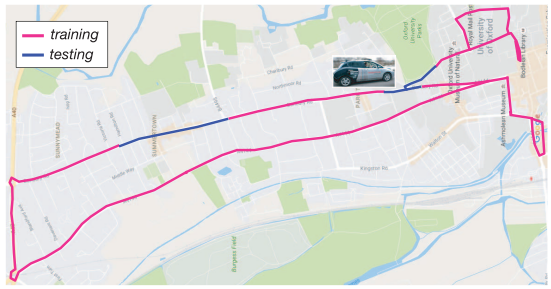

Fig. 9. Training and Testing set collected in a busy urban environment and used for evaluating the importance of the Spatial Transformer when tracking from a moving platform.

7.2.1. Dataset. The evaluation dataset spans 27 km and 85 min of driving. It was collected in a busy urban environment and split 80/20 into training and testing sets with no overlap in geographic locations as shown in Figure 9. The data were collected from a vehicle equipped with two Velodyne HDL64E lasers, resulting in a 360° field of view. The 3D point clouds were reduced to a 2D scan by considering the range of points within a height of 0.6–1.5 m from the ground, as shown in Figure 10.

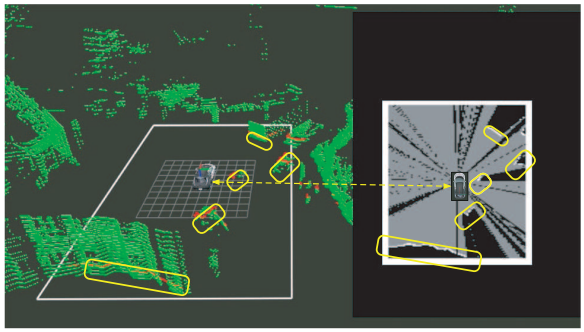

Fig. 10. Conversion of 3D point cloud data into a 2D occupancy and visibility grid as the input to the network. The points considered in the original point cloud range within 0.6–1.5 m vertical from the ground plane local to the robot's location.

7.2.2. Network training. The network was trained on mini-batches of 40 sequences with a frame rate of 10 Hz, alternating between the five inputs shown, and the five inputs hidden. This is a higher frequency than for the static case, and is more appropriate for a moving vehicle given the input field of view of 28×18 m^2 and a vehicle mean velocity of 30 km/h. Longer occluded sequences significantly lower the overlap between the fields of view of the first and final frames of the sequence, since the vehicle is moving through the scene. The GRU3DilConv_48 model was trained on an NVIDIA Tesla K40 GPU until convergence, using the Adagrad stochastic gradient descent optimizer with an initial learning rate of 0.01. We monitor the loss on the validation set to perform early-stopping and prevent over-fitting. Our architecture is implemented on Torch and uses a GPU-based implementation of the *spatial transformer*.[2]

In the static sensor scenario, errors are only backpropagated at observable ground truth locations, i.e. the *visible* part of the space. In the case of a moving sensor, however, we must also consider the fact that as the robot moves in future frames, it will explore new space that falls outside the field of view of the current frame. Consequently, the model should not be falsely penalized for failing to accurately guess objects located within this new space when the input is masked. This is similar in nature to the static case, where the input grid also represents a frontier between what the robot can perceive, and what the model cannot be expected to predict as it is located outside of the sensor horizon. To address this, an additional mask is applied during training on the cost computation and error backpropagation to represent the space that is *predictable* in future frames. Accounting for this drift in the field of view is crucial to achieve good tracking performance, as otherwise, the model is penalized for failing to predict objects outside the sensor horizon. This mask has been overlaid in transparency over the ground truth comparison outputs of Figure 1, and indicates the predictable free space shrinking on future timesteps.

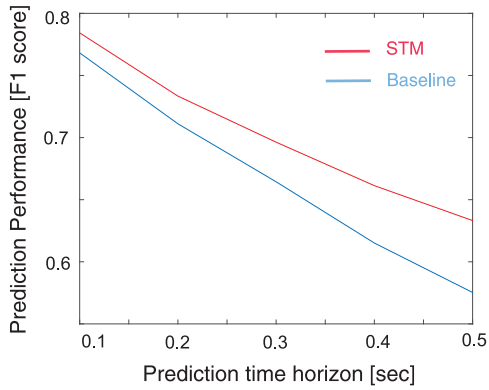

Fig. 11. Positive contribution of the spatial transformer to the network's ability to correctly predict the future occupancy of the scene in a 0.6 s time horizon. The baseline DT does surprisingly well which we attribute to the benign test set where the vehicle mostly evolves at constant velocity down straight roads. We illustrate this hypothesis in Figure 13.

Quantitative results. Figure 11 reports the F1 measure computed with STM and with baseline DT. The baseline DT case utilizes the same architecture as STM with the exception that no ego-motion is taken into account. In other words, the spatial transformer is not used to transform hidden states into the next sensor frame, and no additional mask is applied to the cost computation and backpropagation.

Unsurprisingly, the STM offers improvement over the baseline DT in all future frames, indicating the benefit of incorporating visual ego-motion into the model. Without considering the ego-motion of the vehicle, one might expect a very poor F1 measure for the baseline model. Surprisingly however, it has respectable performance (Figure 11). We suggest three possible explanations for this. Firstly, we posit that this may be due to the relatively benign nature of the dataset: as most of the driving occurs along relatively straight roads at relatively constant velocity ($\sim$ 30km/h) the baseline may have learned a constant velocity model to correct the hidden state update. We illustrate this in Figure 13 discussed in the following section. Secondly, the F1 measure may be dominated by static objects such as walls and buildings, which form a large fraction of the dataset. As such, a respectable F1 score can be attained by learning the vehicle's motion and merely capturing static scenes. Finally, our system may in part suffer from noisy ego-motion estimates accumulated on turns, for which we discuss possible avenues of improvement in Section 8.

Qualitative results. We qualitatively evaluate the model, indicating a selection of sequences where the model performs well, and others where it does more poorly. We also compare qualitatively to the baseline model where no vehicle ego-motion is leveraged. As the frequency and duration of occlusions in the dataset is not as high as the static vehicle dataset, we look at the ability of the network to predict

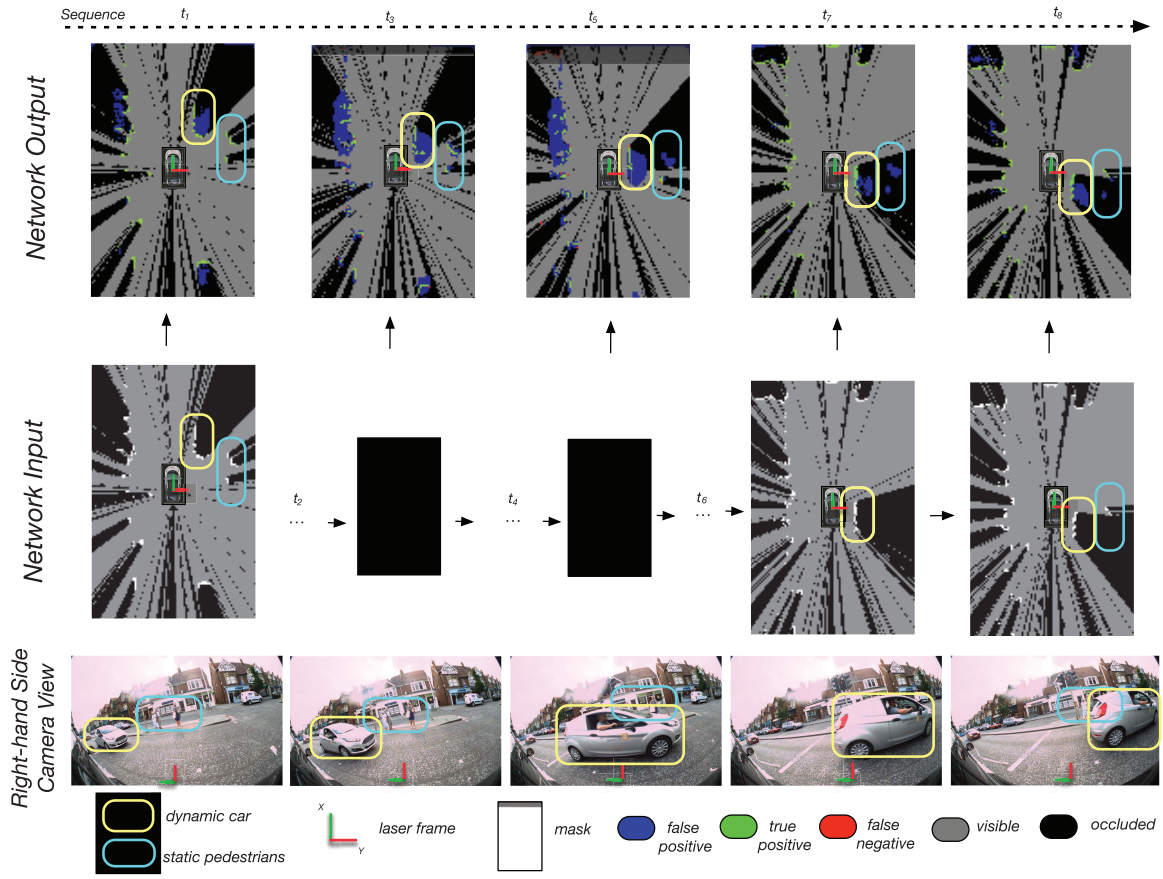

Fig. 12. Left to right, bottom to top: example sequence (0.8 s) of dynamic and static object tracking through occlusion by the STM. The bottom row shows the camera view to the right side of the vehicle for illustrative purposes. The sensor is mounted on the roof of the vehicle which is traveling at a constant speed of 16 km/h throughout the sequence. We highlight the occlusion of static pedestrians (blue contour) caused by a moving vehicle passing between them and the sensor. The occlusion of the moving vehicle (yellow contour) is created by our blanking out of the input from time step $t_{2,\ldots,6}$ included (illustrated by the black network input). The output of the network is evaluated against the visible ground truth. We overlap the mask applied to the cost calculation on the network output grids to highlight the shrinking of the field of view in memory caused by the vehicle ego-motion.

what happens in occlusion by masking the input for five frames out of ten at test time.

Figure 12 illustrates an example of STM accurately tracking both dynamic and static objects through occlusion. Specifically, the model accurately translates the position of the occluded static pedestrians, as well as accurately capturing the dynamics of the vehicle that passes between the pedestrians and the sensor. The occlusion of the car is here simulated by blanking out the input for five subsequent frames.

Figure 13 compares the tracking performance of the STM to that of the baseline model which does not consider the vehicle ego-motion. The five-frame sequence represents full prediction of the scene, as the input has been blacked out to simulate total occlusion. The vehicle is driving at a constant velocity of 45 km/h. Highlighted with a pink rectangle is a moving vehicle passing by, and the tracking accuracy is compared to that of the visible ground truth. Although the STM model achieves very satisfactory tracking of the moving vehicle (bottom output row), we find that the baseline model gradually fails to produce any prediction overlap with the actual position of the passing vehicle at the last frame (top output row). The static background to the left of the scene is however accurately tracked in both models. This sequence suggests that the baseline model may be translating the scene with respect to a learned vehicle ego-motion. As the dataset is over-weighted by static background, we hypothesize that this may be one explanation for the high F1 measure observed for the baseline model.

7.3. Semantic classification through inductive transfer

To solve the problem of estimating both occlusion-free occupancy and scene semantics $\mathbf{h}_t, \mathbf{c}_t$ we consider the static sensor dataset. We hand-label 1000 scans from the training

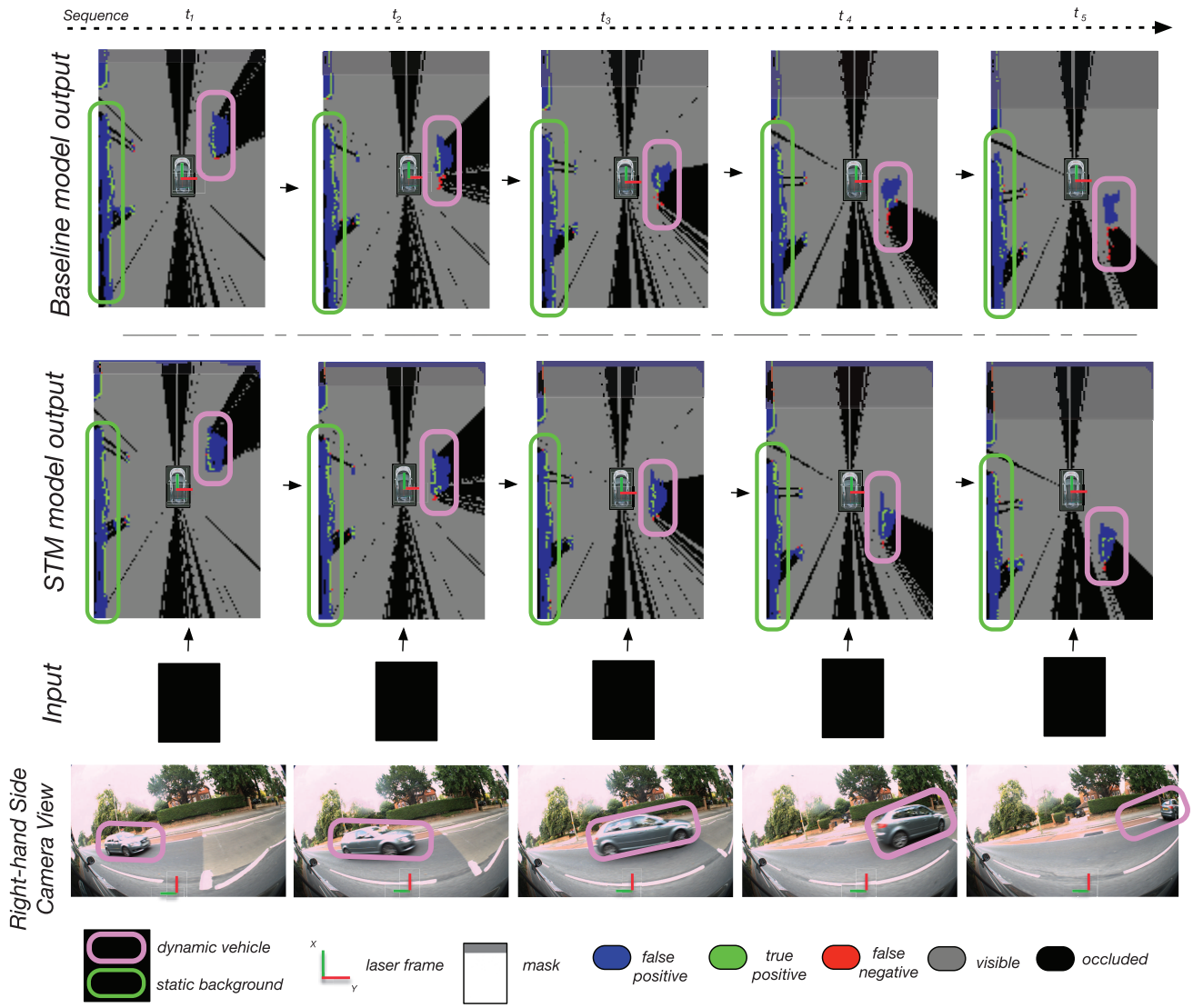

Fig. 13. Left to right: output sequence (0.6 s) of dynamic and static object tracking through occlusion as predicted by the STM (bottom) and the baseline model (top). In this sequence, the sensor's ego-motion is relatively constant and linear. The input to the network is blacked out for the entire sequence shown and simulates total occlusion of the scene. The bottom row shows the camera view to the right side of the vehicle for illustrative purposes. We show the color-coded comparison to the visible ground truth, with false negatives in red, false positives in blue, and true positives in green. Where the spatial transformer network accurately tracks a passing vehicle (pink contour), the baseline model fails to capture the dynamics of the vehicle. This is particularly visible on the last frame, with most of the area around the actual location of these objects showing false negative occupancy prediction for the baseline model. Both models accurately capture the position of the static buildings to the left of the vehicle. This suggests that the baseline model may have learned one of several vehicle ego-motions, and highlights the positive contribution of the STM module in the architecture. Lastly, we overlay the mask applied to the cost calculation on the output grids to highlight the shrinking of the predictable free space as a result of the sensor's ego-motion. As expected, we find that the masked areas correspond to the regions in which the model fails to predict the static background (false negative).

set into four classes for the purpose of network semantic training, and 400 scans from the test set for the evaluation of its semantic classification accuracy. We consider two classification tasks. When performing multi-class semantic prediction, the classes considered are: *pedestrian*, *vehicle*, *cyclist*, and *static obstacle*. In this task, the cyclist and pedestrian classes also contain a few examples of temporarily static members. When performing binary classification, we consider a static object class, and a dynamic object class. In this classification problem, temporarily static pedestrians and cyclists are considered members of the static class along with the permanently static background.

7.3.1. Training procedure. We train a classifier using the same training procedure as the unsupervised tracking task. The training process is illustrated in Figure 2. This is to ensure that the model is forced to utilize the memory and be able to predict the semantics of objects in occlusion. However, unlike the unsupervised task, the classification errors corresponding to the visible part of the scene at $\mathbf{x}_{t+n}$ are *only* back-propagated through the classifier's decoder layer. This is unlike the unsupervised tracking task where errors were back-propagated in time through the hidden states. Models are trained until convergence using the 3 min long labeled dataset, and we monitor the loss on the validation set to perform early stopping and prevent over-fitting. The Adagrad optimizer is used for stochastic gradient descent with an initial learning rate parameter of 0.01. Because the dataset is skewed and contains many more objects of a particular type, e.g. pedestrians, we used a weighting scheme assigning class weights equal to the inverse of class frequency. We do not consider any class weighting when training on the binary classification task (static vs dynamic).

In order to assess the contribution of static biases in the task of semantically labeling the scene, we consider two of our best tracking models (GRU3DilConv_48), as well as its corresponding full network which additionally contains static biases (GRU3DilConvBias_48). Both networks contain GRUs, dilated convolutions, and consider the full hidden state for decoding to the output.

Similarly, to assess the contribution of decoding the full hidden state vs only the 16 layers of the last hidden unit, we compare our two full models (GRU3DilConvBias_48 and GRU3DilConvBias_16) on the task of semantically labeling the scene. Both networks contain GRUs, dilated convolutions, and static biases, and the training procedure is the same as previously described. All models are first pre-trained on the task of predicting the unoccluded occupancy scene on the previously described dataset.

7.3.2. Static vs dynamic objects.

Quantitative results. To quantify the network's ability to classify scene semantics we first consider a binary classification of the predicted output into dynamic and static classes. We compare the F1 measure for the two models (GRU3DilConvBias_48 and GRU3DilConv_48) with a 0.5 threshold for predicting dynamic objects. We find that both models are commensurate in the binary prediction of static vs dynamic obstacles. The full model with static biases predicts the dynamic objects with a mean F1 score of 0.92 while the model without static biases performs with an F1 score of 0.91.

Qualitative results. Figure 14 illustrates a 6 s sequence of binary classification prediction (static vs dynamic) for both models, along with the visible ground truth (Figure 14(a)). As can been seen, both models accurately capture the overall dynamics of the scene, consistently predicting the static background and moving pedestrians on the pavement and road. Circled in red are two temporarily *static* pedestrians which both models are able to capture. However, it remains a challenge for both models to consistently label temporarily static obstacles such as those illustrated by the unclear classification of the dashed orange circles. We find that the full model is able to utilize its dynamic memory to capture temporarily static objects despite having the ability to rely on its static memory to predict the permanent static background. This is supported by the high probabilities of cell occupancy provided by the static biases in Figure 7. We hypothesize that the accuracy of classification of temporary static objects would further improve for both models with more such examples in the data.

7.3.3. Multiple object classes.

Quantitative results. To quantify the network's ability to classify scene semantics, we compute the intersection over union (IoU) for individual and combined classes, as well as compute the confusion matrices for three of our best models (GRU3DilConvBias_48, GRU3DilConvBias_16, and GRU3DilConv_48). These models differ by their use or lack of static biases, and by their decoding of either the last unit, or the whole of the hidden units. Results for class IoU and class confusion matrices are respectively shown in Table 1 and Figure 17. As can be seen, GRU3DilConvBias_48, which contains static biases and decodes the entire hidden state, produces the best classification for all classes considered, reaching an overall classification performance of 0.93 (Table 1(a)).

We further investigate the confusion of classes by observing the recall of classification illustrated in the row normalized table of Figure 17(a). The lowest precision, illustrated in the column normalized table of Figure 17(a), additionally suggests that one third (31.8%) of the model's cyclist predictions correspond to pedestrians. This is a confusion that appears in all models pre-trained on the tracking task (Figure 17(a) to (c) and Table 1(a) to (c)). We suggest this is because in addition to exhibiting similar shapes in 2D laser data, cyclists sometimes stop or cycle more slowly, thus exhibiting the behavior of a pedestrian. Inversely, some pedestrians were observed to run across the scene in both training and testing sets, exhibiting the velocity of a cyclist more than that of a strolling pedestrian.

Our second best model (GRU3DilConv_48), which does not contain static memory, achieves lower performance on all classes (Table 1). The background class particularly achieves a lower IoU metric of 0.82 (vs 0.90). We further observe enhanced confusion between the static background class and respectively the pedestrian and cyclist classes as suggested by the confusion matrices. We hypothesize that the static biases inform the classifier that cyclists evolve on the road, whereas pedestrians are mainly on the pavement and away from static obstacles. These results

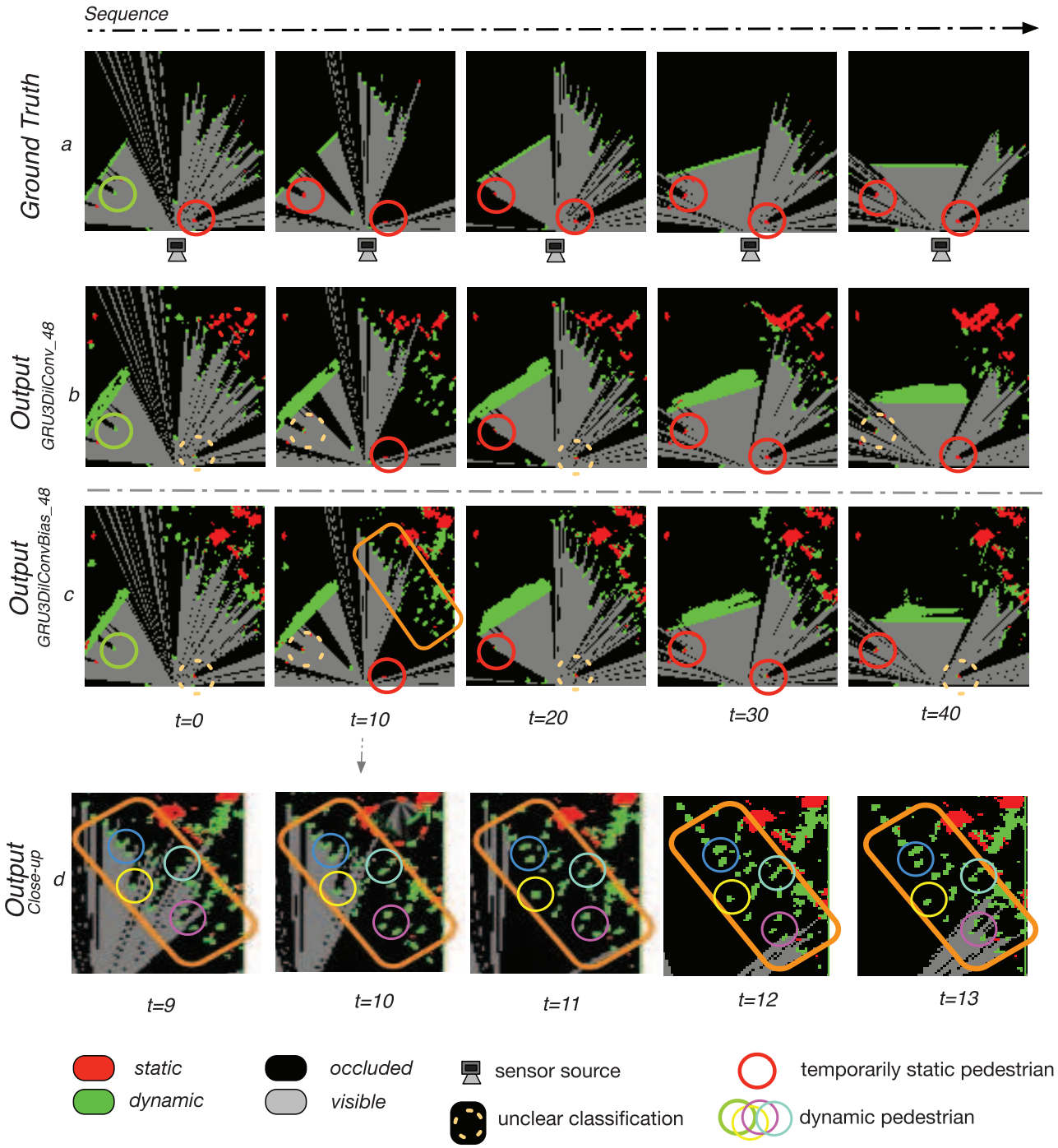

Fig. 14. Example sequence (6 s) of binary classification output (static vs dynamic) for the full model (GRUDilConvBias_48, (c)) and best model (GRU3DilConv_48, (b)). Red labels refer to static obstacles while green illustrates dynamic obstacles. The ground truth is shown for comparison (a). Both models capture the static background and motion of pedestrians in the scene (orange contour close-up, (d)). The red circles show examples of two temporarily static pedestrians which both models capture well. The dashed orange circles represent unclear classifications of the static pedestrians which reflects the difficulty of capturing momentarily static objects.

suggest a positive contribution of static biases to scene understanding.

Performance of the full model when only considering the last output unit in the decoder (GRU3DilConvBias_16) is significantly lower than when considering the full hidden state (GRU3DilConvBias_48), with an overall IoU metric of 0.81. We find that there is more confusion between the pedestrian, cyclist, and vehicle classes. We suggest that the three hidden layers learn to recognize objects of different sizes and motion patterns, consistent with their different

Table 1. Intersection over union (IoU) for individual classes, the combined pedestrian and cyclist classes, and all classes combined, when considering the GRU3DilConvBias model. As explicited by the confusion matrices of Figure 17 (top matrices), the main confusion lies between the cyclist and pedestrian classes as they exhibit similar shapes and velocities in 2D laser data.

Class	Background	Pedestrian	Cyclist	Vehicle	Global
(a) GRU3DilConvBias_48	0.90	0.94	0.58	0.98	0.93
(b) GRU3DilConv_48	0.82	0.91	0.55	0.97	0.89
(c) GRU3DilConvBias_16	0.85	0.84	0.35	0.86	0.81
(d) GRU3DilConvBias_48 (no pre-training on tracking)	0.93	0.66	0.1	0.66	0.62

receptive fields. Although information gets passed forward through the units of the hidden state, we hypothesize that allowing the network to directly access information from the varying scales of the input, may assist positively in discriminating between the semantic classes.

The three models present some small amount of confusion between the static background and pedestrian classes, although static biases contribute favorably in discriminating between the two. As pedestrians walk very close and through the static background on the pavement (mostly shops and buildings), we suggest that these errors are very sensitive to mis-labeling the ground truth.

Qualitative results. A typical input sequence and the corresponding predicted network output as given by the best tracking network (GRU3DilConv_48) and full model (GRU3DilConvBias_48) are shown in Figure 15. The network is able to uncover the unoccluded scene occupancy $\mathbf{y}_t$ and object labels $\mathbf{c}_t$. Despite a long occlusion by a turning bus, the full model is particularly able to retain the position and presence of the occluded pedestrians and static background. The four classes in the scene are accurately captured by both models.

Moreover, it is able to update the positions of dynamic objects through temporary occlusion demonstrating that it has learned to track and recognize objects in the scene.

7.3.4. Value of inductive transfer. We verify the value of the proposed inductive transfer of knowledge by comparing these results to that of learning the semantics of the scene when the model *has not* been pre-trained on the task of tracking. Results can be seen in Figure 17(d) and Table 1(d), and are qualitatively illustrated in Figure 16. The accuracy of the system only achieves an overall IoU metric of 0.62 due to high confusion between the dynamic classes. The permanent static background and buses, characterized by long segments in the data, are however well captured. This is reflected in high values for the row-normalized matrix which suggests that 96.9% and 85.0% of these two classes are correctly classified. Pedestrians and cyclists are however notably confused with vehicles (10.5% and 18.8% mis-labeling respectively) which constitutes the greatest error in the context of safe autonomous robotics. Finally, this model was significantly slower to converge than its pre-trained counterpart ($\times 4$ training epochs). This demonstrates that $\mathbf{h}_t$ offers a powerful semantic descriptor of the scene and can be used as an input for accurate semantic classification.

8. Discussion

The evaluation presented in this paper suggests a positive contribution of GRUs, dilated convolutions and static biases to the more traditional recurrent architecture used in the original deep tracking framework. The most notable step change in terms of improvement on the tracking task is observed when using gated recurrent hidden units. Similarly to the empirical evaluation of Chung et al. (2014), we find that our GRU3_16 model converges faster to a better solution than RNN48. We do not, however, observe any issue related to stability or overfitting of the validation set.

Further model architectures could be investigated. We observe that although the static memory negligibly improves the tracking task, it aids semantic labeling. It would be interesting to find how to best utilize this static memory for learning place-specific information for both tasks. We further hypothesize that in the case of a moving vehicle, static memory could learn something useful about its near surroundings. Tangentially, an interesting follow-up investigation could look into whether the supervised task of decoding the scene semantics might in return improve the tracking task. Unlike typical convolutional networks this network *does not* feature max pooling and maintains the same resolution in each layer. Max-pooling reduces the computational and memory requirements by reducing the size of feature maps. Although this is a desired characteristic for robotics application, we suggest that dilated convolutions provide additional freedom for the network to learn which features to select, where max-pooling would only select the most prominent ones. We observe a forward pass of less than 15 ms on a GPU which does not negatively impact a real-time deployment. However, the effect of max-pooling remains an interesting future model architecture investigation.

Our current implementation of tracking under ego-motion considers a 2D scan constructed from a section of a sparse 3D point cloud. Adding RGB information from a camera would increase the amount of scene information and context available to the tracking and semantic tasks and could supplant the need for static biases. This would move

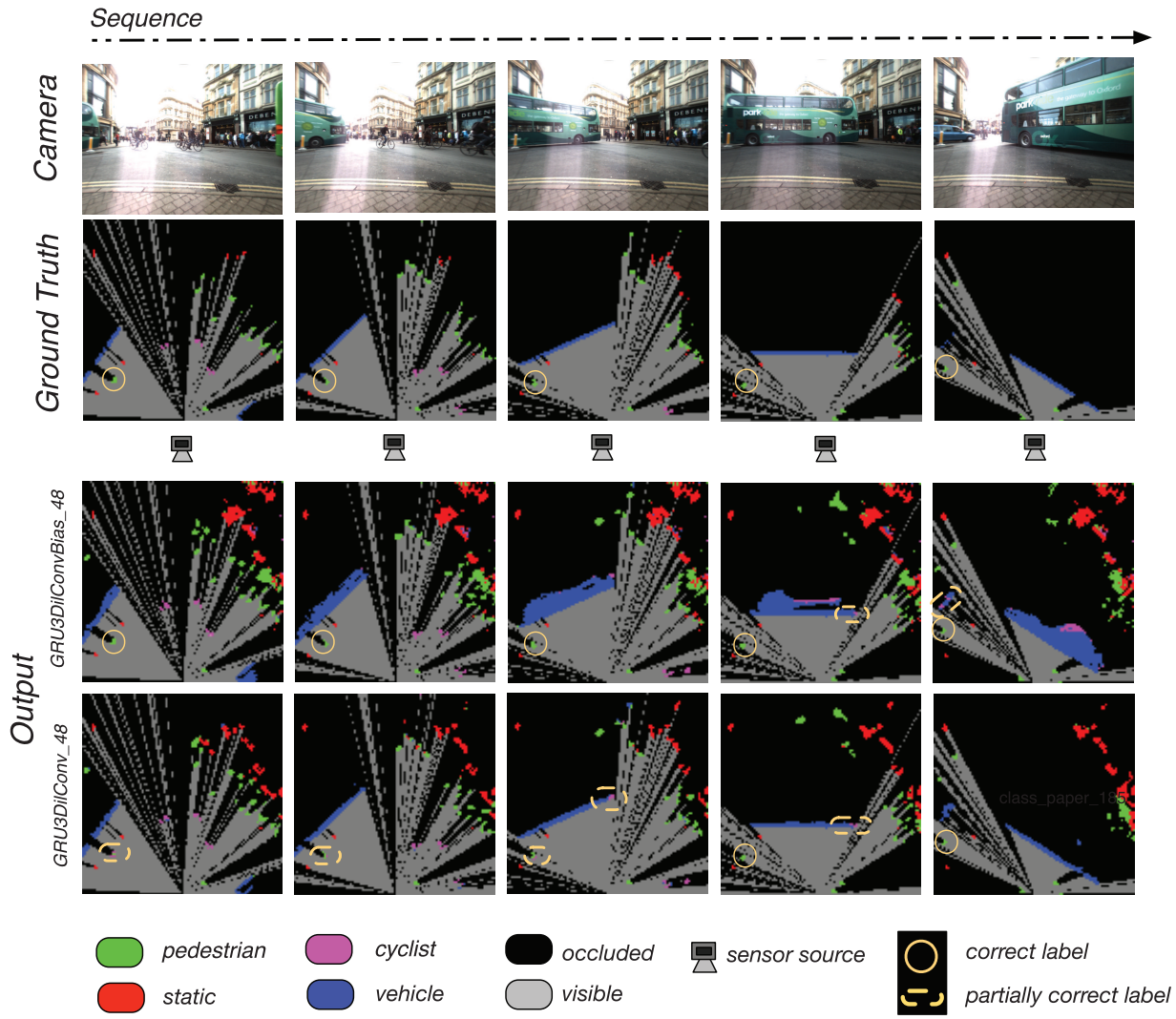

Fig. 15. Example sequence (6 s) of scene classification as produced by GRU3DilConvBias_48 and GRU3DilConv_48 along with the available ground truth camera and laser (top two rows). Despite a long and full occlusion by the turning bus, both models retain an accurate understanding of the scene. A pedestrian is partially and temporarily confused as a cyclist in the GRU3DilConv_48 model (orange circle). This pedestrian is crossing the road at a traffic light (not visible in the camera view) suggesting that the model with static memory may have a prior for pedestrian presence at this crossing. Both models present very partial and limited labelling of the passing vehicles as cyclists.

our work closer to next video frame prediction (Lotter et al., 2016; Patraucean et al., 2015). Alternatively, a multi-modal approach combining both modalities could maintain the current objective of predicting near-future scene occupancy while reducing the amount of network architecture required. An efficient and more favorable extension foreseen would consider the initial 3D point cloud as a means of increasing spatial context to assist the tracking and semantic classification tasks. Sparse convolutions similar to the work by Wang and Posner (2015) and Engelcke et al. (2016) could then be leveraged to maintain computational efficiency.

9. Conclusion and future work

In this paper, we have proposed an approach to perform object tracking for a mobile robot traveling in crowded urban environments. Crucially, unlike classical techniques which employ a multi-stage pipeline, this approach is learned end-to-end with limited architectural choices. By employing a STM, the model is able to exploit noisy estimates of visual ego-motion as a proxy for true vehicle motion. Experimental results demonstrate that *deep tracking* performs favorably to a more traditional model-free tracker in terms of accurately predicting future states, and shows that the model can capture the location and motion of cars, pedestrians, cyclists, and buses, even when in complete occlusion. Further, by inductive transfer of the latent representations learned by the model from the tracking task, it can be applied, with little effort, to the task of semantic classification of surrounding objects.

Future work will look to estimate ego-motion for improved tracking performance in the context of a moving

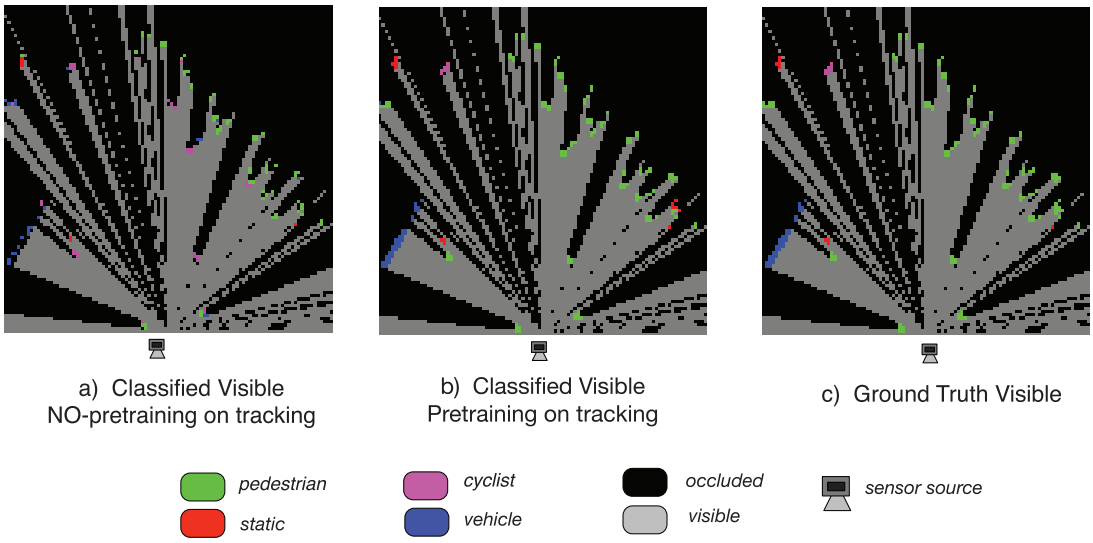

Fig. 16. Visible scene semantics as classified by the full model (GRU3DilConvBias_48) without pre-training on tracking (a) by the full model with pre-training on tracking (b) and by the available visible ground truth (c). For a fair comparison that does not require tracking through occlusion, we only show the predicted labels of the visible occupancy grid. To the exception of the incoming bus and part of the static background which are accurately labeled by the untrained network, the scene demonstrates the powerful semantic descriptor offered by the $\mathbf{h}_t$.

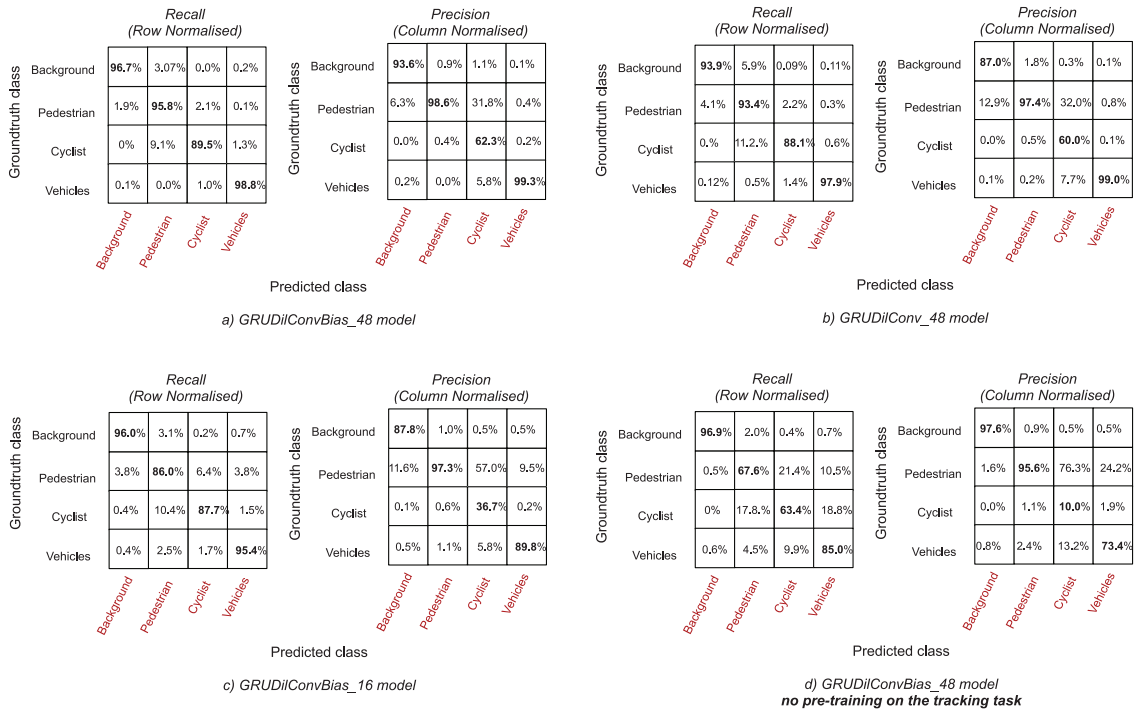

Fig. 17. The confusion matrices of the semantic classification performance. (a) GRU3DilConvBias_48 represents the full model and decodes the entire hidden state. This model achieves the highest classification results. (b) GRU3DilConv_48 decodes the entire hidden state but does not consider static memory. This model is less able to differentiate the dynamic classes from the static background. (c) GRU3DilConvBias_16 decodes only the last 16 hidden layers. This model is less able to differentiate cyclists and pedestrians from the other classes which suggests that high resolution information may be diluted in the higher layers of the network. (d) GRU3BiasDilConv_48 decodes the full hidden state but was not been trained to capture the scene dynamics in a prior task of tracking. The static biases appear to positively contribute to classifying the static background but the dynamic classes suffer high confusion.

platform, explore modalities such as radar, and extend the approach to 3D. Another interesting extension would be to allow the STM to be learned or fine-tuned during the training process, to correct errors in the estimated ego-motion. We release our dataset for further use within the community.[3]

Acknowledgements

The authors would like to gratefully acknowledge support of this work by the UK Engineering and Physical Sciences Research Council (EPSRC) Doctoral Training Programme (DTP) and Programme Grant DFR-01420, as well as the Advanced Research Computing (ARC) services at the University of Oxford for providing access to their computing cluster.

Notes

1. https://github.com/pondruska/DeepTracking
2. https://github.com/qassemoquab/stnbhwd/
3. Dataset available at: www.ori.robots.ox.ac.uk

References

Arras K, Mozos O and Burgard W (2007) Using boosted features for the detection of people in 2D range data. In: *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), Roma, Italy, April 10–14, 2007*. New York, NY, USA: IEEE, pp. 3402–3407.

Byravan A and Fox D (2016) Se3-nets: Learning rigid body motion using deep neural networks. *arXiv preprint arXiv:1606.02378, 2016* .

Caruana R (1995) Learning many related tasks at the same time with backpropagation. In: *Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA, November 27-December 2, 1995*, volume 7. Cambridge, MA, USA: The MIT Press, pp. 657–664.

Cho K, Van Merriënboer B, Gulcehre C, Bahdanau D, Bougares F, Schwenk H and Bengio Y (2014) Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078, 2014* .

Choi S, Lee K and Oh S (2016) Robust modeling and prediction in dynamic environments using recurrent flow networks. In: *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Daejeon, Korea, October 9–14, 2016*. New York, NY, USA: IEEE, pp. 1737–1742.

Chung J, Gulcehre C, Cho K and Bengio Y (2014) Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555, 2014* .

Dahl GE, Yu D, Deng L and Acero A (2012) Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. In: *IEEE Transactions on Audio, Speech, and Language Processing, 2012*, volume 20. New York, NY, USA: IEEE, pp. 30–42.

Dequaire J, Rao D, Ondruska P, Wang D and Posner I (2016) Deep tracking on the move: Learning to track the world from a moving vehicle using recurrent neural networks. *arXiv preprint arXiv:1609.09365, 2016* .

Engelcke M, Rao D, Wang DZ, Tong CH and Posner I (2016) Vote3deep: Fast object detection in 3D point clouds using efficient convolutional neural networks. *arXiv preprint arXiv:1609.06666, 2016* .

Hochreiter S and Schmidhuber J (1997) Long short-term memory. *Neural computation* 9(8): 1735–1780.

Jaderberg M, Simonyan K, Zisserman A and Kavukcuoglu K (2015) Spatial transformer networks. In: *Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, December 7–12, 2015*. Red Hook, NY, USA: Curran Associates Inc, pp. 2017–2025.

Krizhevsky A, Sutskever I and Hinton GE (2012) Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems (NIPS), Lake Tahoe, NV, USA, December 3–6, 2012*. Cambridge, MA, USA: The MIT Press, pp. 1097–1105.

Le QV (2013) Building high-level features using large scale unsupervised learning. In: *Proceedings of the IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), Vancouver, BC, Canada, May 26–31, 2013*. New York, NY, USA: IEEE, pp. 8595–8598.

Lotter W, Kreiman G and Cox D (2016) Deep predictive coding networks for video prediction and unsupervised learning. *arXiv preprint arXiv:1605.08104, 2016* .

Medsker L and Jain L (2001) Recurrent neural networks. In: *Design and Applications, 2001*, volume 5. CRC press.

Mitchell T and Thrun S (1993) Explanation-based neural network learning for robot control. In: *Advances in Neural Information Processing Systems (NIPS), Denver, CO, USA, 1993*, volume 5. San Mateo, CA, USA: Morgan Kaufmann.

Ondrúška P, Dequaire J, Wang DZ and Posner I (2016) End-to-end tracking and semantic segmentation using recurrent neural networks. *arXiv preprint arXiv:1604.05091, 2016* .

Ondrúška P and Posner I (2016) Deep tracking: Seeing beyond seeing using recurrent neural networks. In: *Proceedings of the Association for the Advancement of Artificial Intelligence (AAAI), Phoenix, AZ, USA, February 12-17, 2016*. Palo Alto, CA, USA: AAAI Press.

Pan SJ and Yang Q (2010) A survey on transfer learning. In: *IEEE Transactions on Knowledge and Data Engineering*, volume 22. Los Alamitos, CA, USA: IEEE Computer Society, pp. 1345–1359.

Pascanu R, Mikolov T and Bengio Y (2012) On the difficulty of training recurrent neural networks. *arXiv preprint arXiv:1211.5063, 2012* .

Patraucean V, Handa A and Cipolla R (2015) Spatio-temporal video autoencoder with differentiable memory. *arXiv preprint arXiv:1511.06309, 2015* .

Pennington J, Socher R and Manning CD (2014) Glove: Global vectors for word representation. *Proceedings of the Empiricial Methods in Natural Language Processing (EMNLP 2014)* 12: 1532–1543.

Petrovskaya A and Thrun S (2009) Model based vehicle detection and tracking for autonomous urban driving. *Autonomous Robots* 26(2): 123–139.

Vu TD, Aycard O and Appenrodt N (2007) Online localization and mapping with moving object tracking in dynamic outdoor environments. In: *Proceedings of the IEEE Intelligent Vehicles Symposium, Istanbul, Turkey, June 13–15, 2007*. Red Hook, NY, USA: Curran Associates, Inc, pp. 190–195.

Wang DZ and Posner I (2015) Voting for voting in online point cloud object detection. In: *Proceedings of the Conference for Robotics: Science and Systems (RSS), Rome, Italy, July 13–17, 2015*. Cambridge, MA, USA: The MIT Press.

Wang DZ, Posner I and Newman P (2015) Model-free detection and tracking of dynamic objects with 2D lidar. *The International Journal of Robotics Research* 34(7): 1039–1063.

Wang T, Wu DJ, Coates A and Ng AY (2012) End-to-end text recognition with convolutional neural networks. In: *Proceedings of the International Conference on Pattern Recognition (ICPR), Tsukuba Science City, Japan, November 11–15, 2012*. Red Hook, NY, USA: Curran Associates Inc, pp. 3304–3308.

Xingjian S, Chen Z, Wang H, Yeung DY, Wong Wk and WOO Wc (2015) Convolutional LSTM network: A machine learning approach for precipitation nowcasting. In: *Advances in Neural Information Processing Systems (NIPS), Montreal, Canada, December 7–12, 2015*. Red Hook, NY, USA: Curran Associates Inc, pp. 802–810.

Yang SW and Wang CC (2011) Simultaneous egomotion estimation, segmentation, and moving object detection. *Journal of Field Robotics* 28(4): 565–588.

Yu F and Koltun V (2015) Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122, 2015*.

Zhao L and Thorpe C (1998) Qualitative and quantitative car tracking from a range image sequence. In: *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Santa Barbara, CA, USA, June 23–25, 1998*. Washington, DC, USA: IEEE Computer Society, pp. 496–501.