Understanding Future Motion of Agents in Dynamic Scene Using Deep Learning

by

Anuj Sharma

prepared at

Torr Vision Group
Department of Engineering Science
University of Oxford

submitted to

University of Evry-Val-d'Essonne (UEVE), Evry
& University of Paris Saclay, Paris
(August 2017)

in partial fulfillment of

Masters in Smart Aerospace and Autonomous Systems (M2-SAAS)

supervised by:

Prof. Philip H. S. Torr
Dr. Puneet K. Dokania

Keywords: Future trajectory prediction, Attention-Mechanism, Scene Context Fusion, Vehicle Dynamics, Interactions, CNN, RNN-Encoder-Decoder

I dedicate my thesis to my family and friends for their endless support.

Abstract

Autonomous vehicles garner huge importance, subject to their promise of safe driving, thereby preventing the loss of so many lives that happen every year in road accidents. However, it is yet a bit far in the future to be able to develop a full fledged autonomous vehicle that can handle any complex traffic situation anywhere in the world. This is due to the many challenging problems associated with building an autonomous car, few of them are – (i) identification and understanding of roads, traffic signals and traffic rules; (ii) detection of agents in the scene (vehicles, cyclists, pedestrians etc.); (iii) understanding the behavior of different agents in the scene; and (iv) trajectory planning.

Many of the above mentioned tasks are yet to be solved properly in order to use them in autonomous cars. One among them is behavior understanding of the agents in the scene. This involves understanding how the agents (vehicles/pedestrians/cyclists etc) will behave short time in the future given the current situation. Precisely understanding the near future behavior of other agents in the scene will help the autonomous car to better plan its trajectory and also to handle many unexpected situations. Based on these motivations, we build a model that can learn to understand the motion characteristics of agents in the scene and thus allows us to predict their future. The problem can be viewed as a task of relating the past to future motions conditioned on how an agent moves subject to its own motion characteristics, scene (roads, sidewalks, etc), and its interactions with neighboring agents.

In more detail, our model takes the past trajectory and the scene context (image of the scene) as inputs to predict the future trajectory of the agent (vehicle/pedestrian/cyclist etc) under consideration. It consists of following main components – (1) RNN-ED (Recurrent Neural Network Encoder Decoder); (2) SCF (Scene context fusion); and (3) AM (Attention mechanism).

The first component, RNN-ED, is used for the time series prediction. The basic idea behind RNN-ED is to use RNN to encode the given input series into a latent space and then use the encoded representation to decode, using another RNN, into the future/output series. The role of the second component, SCF, is to fuse the scene information and the interactions among the nearby agents with the trajectory information of the agent under consideration. In more detail, we project the input image into a feature space learned using Convolutional Neural Network (CNN) in order to capture scene context. Similarly, the input trajectories of the agents in the scene is mapped into another feature space to model interactions among them. Finally, these feature vectors are used together to fuse the scene and the interaction together to obtain a richer feature representation. The last and an important component, AM, of our model intelligently weighs all the past summaries (hidden state vectors) w.r.t. the future scenario and *attends or weights* to the past summaries depending on their resemblance to the future scenario. This helps the model to find the important past patterns useful to decide the future trajectory.

To demonstrate the efficacy of our model, we use a highly challenging real-world traffic scenario dataset, Stanford Drone Dataset. This dataset contains many highly dynamic situations (cross-roads/roundabouts etc.) with many agents (cars/pedestrians/cyclists etc.) in many different dynamics (slow/fast moving, sharp maneuver, static, etc.).

Acknowledgements

First and foremost, I would like to sincerely thank my supervisor Prof. Philip H S Torr for providing me with the esteemed opportunity to carry out my Masters Thesis under his supervision. I would also like to extend my thanks to Dr. Puneet K. Dokania and Namhoon Lee for their constant guidance and supervision throughout the work.

Special thanks to my post-graduate school University of Evry-Val-d'Essonne (UEVE), Evry and University of Paris-Saclay, Paris, Professor Yasmina Bestaoui Sebbane and Professor Dalil Ichalal for their continued support throughout the duration of my work at the University of Oxford.

Finally, I would like to thank my wife, my parents and my brother for their love and unconditional support throughout my Masters.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Overview

Autonomous vehicles are one of the most challenging technologies currently being worked on. Several institutes, industries (Google, Uber, Tesla etc.) and governments all across the world are working on it to make it a reality. Due to the loss of so many lives every year in road accidents, the realization of this concept has become even more essential. For humans, learning to drive is intuitive. But to develop a vehicle which could drive autonomously like humans is highly challenging. The concept has been around for a long time, first successfully demonstrated in the 1980s [21, 22, 55]. Since then, a great deal of progress has been made in this direction. But as rightly pointed out in [32], we still have a long way to go before we could develop autonomous vehicles that could drive in unknown complex environments.

Driving involves many aspects such as (i) identification of roads, (ii) understanding of road signs, traffic signals, (iii) detection of neighboring vehicles, pedestrians, cyclists etc., (iv) tracking of objects of interest, (v) understanding their motion pattern and (vi) predicting their future trajectories and many more. But its difficult to describe them mathematically using hand engineered models. Thus, people have used recent state of the art 'Deep Learning' [10] to address them. Similar to human brain's neural network, deep learning involves development of artificial neural networks using compositions of functions, whose parameters are learnt through training on some available data. The working principle resembles how our brain learns through experience. Thus, deep learning helps in development of models that are powerful enough to understand the various aspects of driving.

With vision playing a crucial role in driving, the concept of autonomous vehicles has been broken down into smaller vision problems as thoroughly discussed in [32]. It presents the challenges and recent state-of-the-art methods in these problems namely object detection [13, 15, 16, 17], semantic segmentation [14, 20, 27, 40, 44, 47, 56, 64, 65, 66], tracking [11, 39, 60, 61, 62], future prediction [7, 9, 31, 38], reconstruction [26, 41, 51], pose estimation [30, 52], scene understanding [23, 59, 63] and SLAM (Simultaneous Localization and Mapping) [42, 43].

Out of the above mentioned aspects, we have focused on understanding the motion characteristics of agents (vehicles, pedestrians, cyclists etc) and predicting their future motion in a dynamic scene for this work. The problem can be viewed as a task of relating the past to future motions conditioned on how an agent moves subject to its own motion characteristics, scene (roads, sidewalks, etc), and its interactions with neighboring agents.

1.2 Problem Definition

Let us imagine a traffic scenario with dynamic agents. These agents may move on straight or curved paths, take different exits at the roundabouts or exit into some left/right cut etc. At road junctions, some agents may allow others to exit first. Heavy vehicles such as trucks or buses may not make sharp maneuvers compared to smaller vehicles such as cars or motorcyclists. Also, a car may stop for a while to let the pedestrians cross the road first. Thus, it can be seen that each agent has its own motion pattern, which is a combination of dynamics, scene and interactions with nearby agents. Fig. 1.1 presents a traffic scenario taken from [49] in order to understand the complexity of the problem. Note that the agents move subject to the presence of roads, side-walks, roundabout and ensure that they don't collide with each other. Through our model, we aim to understand such complex relationship and predict future motion of agents.

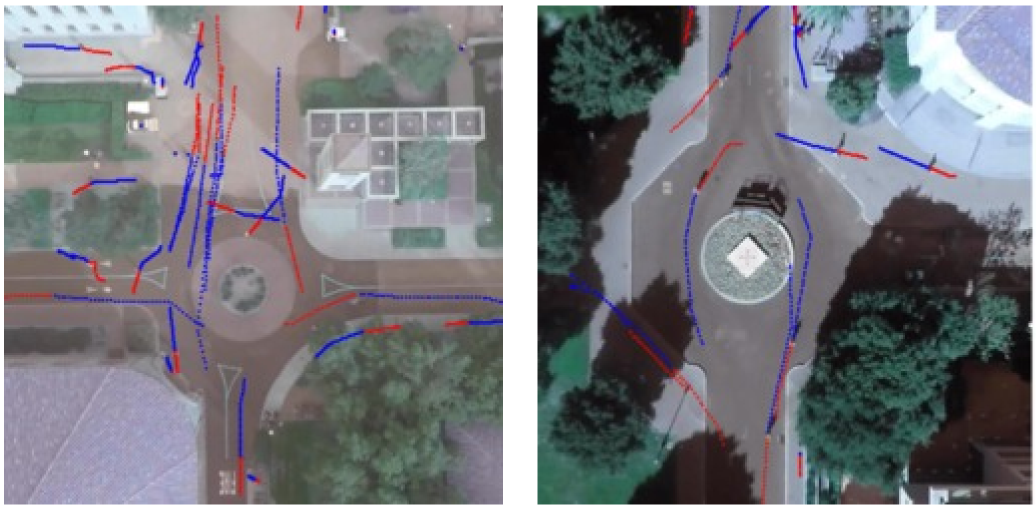


Figure 1.1: Example of real traffic scenario of several agents moving in the scene. Taken from [49]. Red and blue depict past and future motions respectively. Agents can be seen to be moving subject to the scene (road, roundabout, side-walks etc) and ensuring no collisions with each other.

During driving, we analyze the motion characteristics of nearby agents by observing them for some time and then predict their future positions. This helps us in planning our future course of action. We primarily use our vision to analyze this. We only look at how the agents move with time and relate it with scene and extract motion characteristics.

Mathematically, with reference to the above description, we aim to learn a mapping, f, from the past trajectory $\mathcal{X}$ to the future trajectory $\mathcal{Y}$ subject to the scene and interactions, as follows:

$$f_\theta : \mathcal{X} | (scene,\ interactions) \rightarrow \mathcal{Y} \tag{1.1}$$

1.3 Related Work

The above relationship between the past and the future subject to scene context in Eq. 1.1 is highly complex and can't be expressed explicitly using mathematics. An earlier attempt has been made to understand pedestrian dynamics using 'Social Force' model [28]. But its extension to a real driving scenario involving interactions between agents and scene may not be very accurate, since behavior of agents would vary from scenario to scenario. Thus, instead of explicitly hand designing a mathematical model, researchers have addressed this problem by learning the parameters of a family of functions using machine learning, specifically, deep learning [7, 9, 31, 35, 37, 38, 57, 58]. However, these works normally do not consider a dynamic scene with scene and interactions in entirety. A very recent work [38] considers the scene and interactions but the interactions are not dynamic. Also, it looks only at a patch of the entire scene and assumes that the agents outside the patch will not impact the motion of those lying within the patch. This is a big assumption, since in future some of the agents from out of the patch may be close enough to impact the motion of some agent in the patch. Our work is closest to [38] and borrows many inspirations from it.

1.4 Our Approach and Novelties

Taking motivation from Deep Learning, we developed a model that simulates dynamic scene and interactions and predicts the future trajectory given the past trajectory. We utilized two main concepts of Deep Learning namely CNNs (Convolutional Neural Networks) and RNNs (Recurrent Neural Networks). The parameter vector θ in Eq. 1.1 represents the combined parameters of the CNN and RNN used in our model and is learned using the training dataset. We used Encoder-Decoder variant of RNN [8, 18, 38, 46] to model the mapping from $\mathcal{X}$ to $\mathcal{Y}$ as given in Eq. 1.1. The *scene* in Eq. 1.1 is obtained by extracting scene features through CNN as in [38] and *interactions* based on 'Social-Pooling' concept presented in [7].

An important feature of our model is that it is implemented with an 'Attention-Mechanism' [8, 46]. In this, special focus is made to a past situation which most resembles the future scenario. It is expected that an agent would exhibit a similar behavior in similar situations. Hence, it is important to find out if a situation similar to the one in future ever happened in the past. And since the past motion is known, it could be utilized to expect a similar behavior from the agent. Unlike a similar architecture [38], our model has following novelties:

- It can handle variable number of agents per scene
- It is able to simulate dynamic scene i.e. scene features and interactions change with the positions of the agents
- Agents that are far-off may indirectly impact each other's motions by impacting the agents lying in between them. This effect though minimal would still be captured by our model as it focuses on the entire scene instead of a patch of it.

1.5 Experiments and Code

We carried out an end-to-end training of the model on a highly challenging real-world Stanford Drone Dataset (SDD [49]). This dataset contains many highly dynamic situations (cross-roads/roundabouts etc.) with many agents (cars/pedestrians/cyclists etc.) in many different dynamics (slow/fast moving, sharp maneuver, static, etc.).

We develop the entire code base using the well known TensorFlow [6] library. Due to the unavailability of the code for DESIRE [38], that works on a similar setup as ours, we developed the baseline codes ourselves. Also, the data split is not the same as used in DESIRE; hence our results cannot be directly compared to that quoted in DESIRE [38]. We will make our code publicly available in due course of time.

1.6 Observations

From our experiments, we observed that the test performance of our model is comparable to the encoder-decoder based model presented in [38], thereby justifying the usage of 'Attention-

Mechanism' with fusion of dynamically changing agent's dynamics, scene and interactions. Please note that we do not compare with the final model of DESIRE [38] as it is based on a generative model, thus, depends on a family of potential future trajectories. Hence, it can not be directly compared. However, the results, if not better, are very close to the final model of [38].

1.7 Layout

This thesis is laid out in the form of relevant literature, methodology used, experiments, results, conclusions and future scope of work. Chapter 2 explains the architecture and functionalities of Neural Networks, CNNs and RNNs. Chapter 3 discusses the neural network architecture developed for future trajectory prediction using CNN and RNN in detail. Chapter 4 presents the experimental setup, datasets used, learning criteria and results obtained. The concluding remarks and future scope of work are presented in Chapter 5.

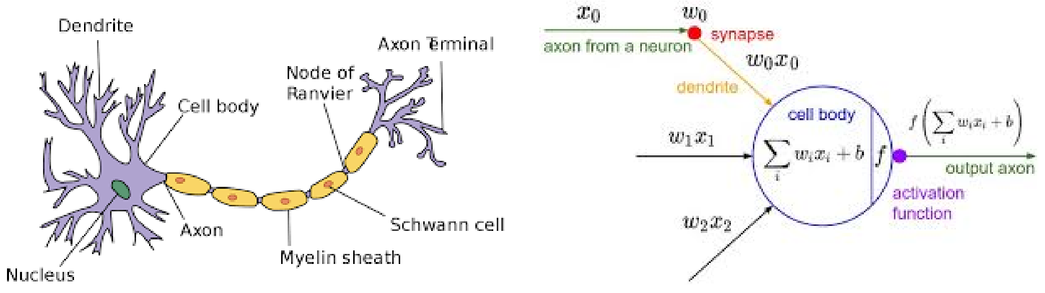
Chapter 2

Background

In this chapter, we present a background of the Neural Networks, Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), which we have utilized in Chapter 3 to build our model for trajectory prediction of agents conditioned on scene and interactions.

2.1 Neural Network

Neural network, a universal function approximator [12], takes its inspiration from the functioning of the human brain. As shown in Fig. 2.1a, our brain is a highly complex network of neurons where each neuron communicates with others in order to process data that we normally perceive through our sense organs. Similarly, a neural network as shown in Fig. 2.2 is a collection of artificial neurons (a simple mathematical function) where each artificial neuron is connected to another in a pre-specified pattern so that the final mathematical function realized using this collection is able to mimic large class of complex functions, and is easier to analyze and understand at the same time.



(a) (b)

Figure 2.1: Representation of biological neuron and its mathematical form. Taken from [3, 1]

A neural network has several layers of neurons viz. input, hidden and output layers. The acquired data is fed to the input layer of the neural network. The input layer processes the data and passes the processed information to the hidden layers. These layers further process and pass onto the output layer which generate the outputs corresponding to the inputs to the neural network. These layers can be very easily related to how we use our vision. For example, when we look at a car in the environment, we are able to quickly identify it as a car instead of identifying it as an animal or a human being. The image that we capture is fed to the *input layer* of our brain. And then instantly we get a decision from the *output layer* of our brain that the object we are looking at is a car. But, we don't make any explicit effort to carry out this

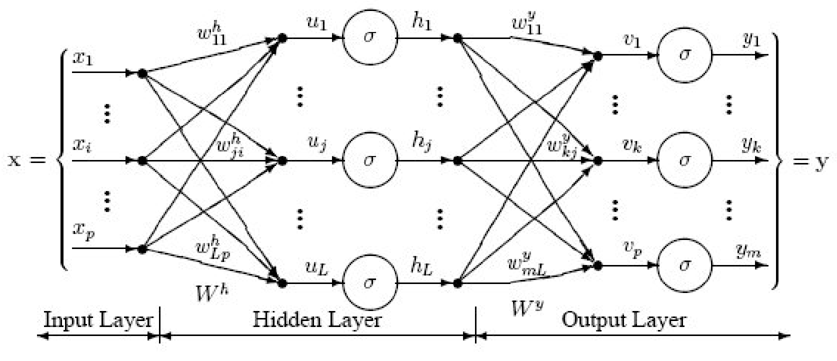


Figure 2.2: Neural Network

analysis. This is done internally by our brain through the *hidden layers.*

2.1.1 Neuron

The functioning of a neuron is mathematically expressed by Eq. 2.1 and shown in Fig. 2.1b. It can be interpreted as computing a weighted sum over all the inputs and then passing it through some non-linear activation function. The weighted sum is computed using the weights W and the biases b. The activation function is given by *f*.

$$y = f(Wx + b) \tag{2.1}$$

2.1.2 Activation Function

Following are the activation functions generally considered.

- **Sigmoid** Sigmoid function (σ) given by Eq. 2.2 is used to map the input x to $y \in$ [0,1]. It is shown in Fig. 2.3a.

$$y = \sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.2}$$

- **Hyperbolic Tangent (tanh)** Hyperbolic tangent (tanh) function is used to map the input x to $y \in$ [-1,1] using Eq. 2.3. It is represented in Fig. 2.3b.

$$y = \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.3}$$

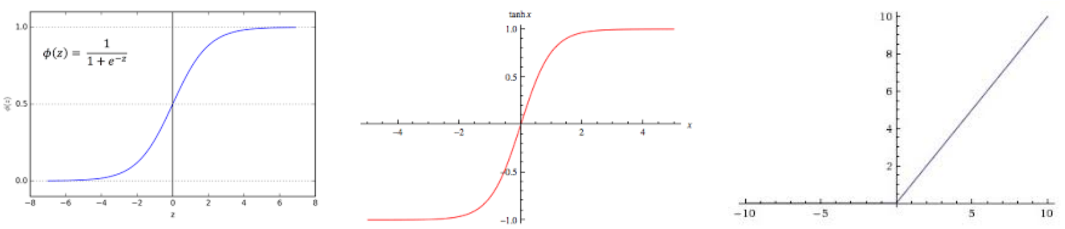
- **Rectified Linear Unit (ReLU)** ReLU outputs the maximum between the input x and zero. It is given by Eq. 2.4, where x and y are respectively the inputs and outputs of ReLU activation. The ReLU function is shown in Fig. 2.3c.

$$y = \max(0, x) \tag{2.4}$$

2.1.3 Training

The objective of the network is to tune its parameters to achieve the given target Y. A forward propagation of the network is carried out over the given inputs X to get an output $\hat{Y}$. The output $\hat{Y}$ is then compared with the given target Y and the error between them is computed through loss function L_p-norm given by Eq. 2.5. Usually, L_1 or L_2 losses are used, wherein $p = 1, 2$ respectively. The loss is back-propagated [48] to get the gradients at each neuron, which are then used to tune the parameters of the model.

$$L_p = ||Y - \hat{Y}||_p \tag{2.5}$$



(a) Sigmoid (b) tanh (c) ReLU

Figure 2.3: Activation functions

2.2 Convolutional Neural Network (CNN)

CNNs are neural networks that are highly utilized in tasks such as classification [36, 50], object detection [13, 15, 16, 17, 33], segmentation [14, 20, 27, 40, 44, 47, 56, 64, 65, 66], hand writing recognition [45] etc. They have high representational capacity and are able to extract relevant features/ information in the inputs.

CNNs use a convolution of filters as a neuron. A neuron in CNN looks at a region of inputs and performs convolutions on them. CNNs usually have multiple layers with each layer consisting of convolution filters and pooling operations. At the end, there are usually *fully-connected* layers that map to the number of classes of objects or outputs under consideration. Normally, the number of depth channels increase and the cross-sectional size of the image decrease with the layers.

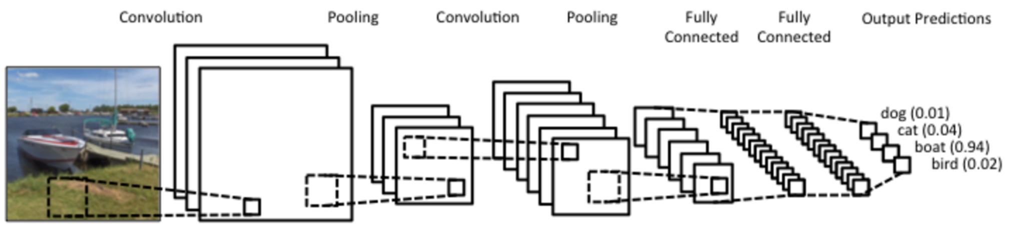


Figure 2.4: CNN for image classification. Taken from [4]

For example, a CNN pipeline for image classification task is shown in Fig. 2.4. Four classes of objects –dog, cat, boat and bird– are considered. The figure shows the usage of layers of convolutions and pooling operations. In the end, *fully-connected* layers are employed. Also it can be seen that the image size is decreasing and the number of channels is increasing as the flow moves further into the CNN pipeline. This gradual decrease in size of the image can be interpreted as attempting to extract relationship between far-off pixels in the original image. If the image size is suddenly reduced, there would be a loss of information. So to preserve it, a gradual reduction is preferred. The pooling operation represents usage of the most prevalent/ average information in a region. The increase in number of channels can be attributed to the fact that they try to capture maximum features in the image. Due to the depth (number of layers) of the network, a long chain between the inputs and the outputs is formed. And there exists a link between how the weights in the first layer affect the end output through a series of layers. The aim of the CNN is to tune its parameters in all the layers subject to the training objectives.

The input image in Fig. 2.4 is a boat. Hence, the CNN should learn to predict that the image corresponds to a boat. The image has 3 channels namely RGB (Red, Green and Blue).

The CNN takes as inputs these 3 channels and in the end outputs a score to each class. The score represents the probability of the image belonging to that class. This CNN is trained using a cross-entropy loss which compares the predicted probability and the actual probabibility of the image being a boat.

The CNN in the above example is able to capture the features present in the image through its deep network. It is able to find a relationship between the pixels and then predict the class of the image. Hence, due to its depth, it has high representational capacity.

CNN is initially trained on a set of images. Then it is tested on a set of similar images and test performance is evaluated.

2.3 Recurrent Neural Network (RNN)

As discussed above, CNNs are powerful enough to identify the features in an image. From their implementation, it is evident that they can only be used on individual images. They cannot be used to find a relationship between a sequence of inputs. Sequential inputs such as sentences, time series data etc cannot be represented using CNNs. For sequential data, we use Recurrent Neural Networks (RNNs) which have the capacity to learn the relationship between sequentially changing data [24, 25, 53].

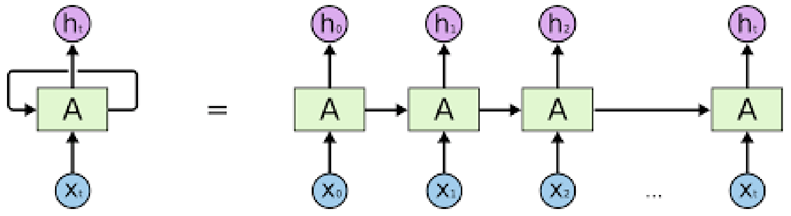
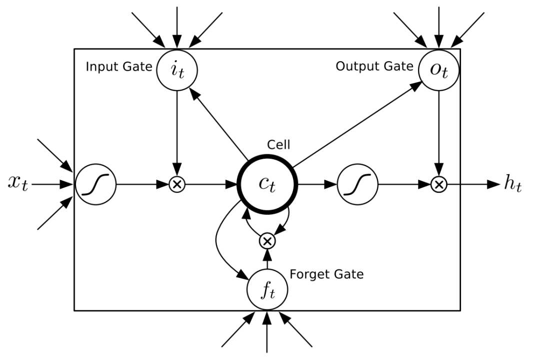


Figure 2.5: Recurrent Neural Network. Taken from Colah's wonderful blog [5]

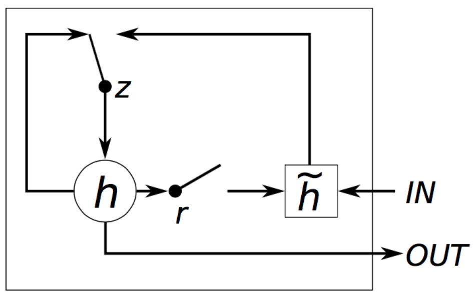
RNNs are neural networks that have a sequence of neurons, with each neuron looking into data at different time steps. All these neurons are sequentially connected. RNN is pictorially represented in Fig. 2.5. It takes a series of inputs $(X_0, X_1, X_2, X_3, \cdots, X_t)$ and learns the features at each time step namely $(h_0, h_1, h_2, \cdots, h_t)$. The box named as A in Fig. 2.5 that links the input to the feature is known as *cell.* The cell has a bunch of neurons which are able to capture the relationship between the inputs upto the current time step and accordingly output a feature representation. The cells at all time steps share the parameters. Usually, cells are selected to be LSTM (Long Short Term Memory [29]) or GRU (Gated Recurrent Unit [18]) as these have been shown to be powerful enough to retain long term memory. They are able to relate to some information that was seen a long time back. Hence, they are utilized for time series predictions [38, 46], long term sequence translations [8, 18, 54] etc.

LSTM and GRU cells are shown in Fig. 2.6. The LSTM (Fig. 2.6a) has 3 neurons namely input gate, forget gate and output gate. The input gate is used to determine how much to update the cell based on the current inputs. The forget gate decides how much of the cell information to be dropped. And the output gate corresponds to how much content of the cell should be used to represent the features at the current time step. Thus, there are two types of states in LSTM –cell state and output state.

On the other hand, GRU (Fig. 2.6b) has 2 neurons namely update and reset gates. The update gate z decides how much to update the hidden state. And the reset gate decides how much of the states to be reset subject to the current inputs. It uses only one set of hidden states unlike LSTM that has two sets of states.



(a) LSTM



(b) GRU

Figure 2.6: RNN Cells - LSTM and GRU. Taken from [2, 18]

From the above, it can be seen that GRU has easier implementation (2 gates, 1 state) in comparison to LSTM (3 gates, 2 states). Also, their performances are shown to be equivalent [19]. Thus, owing to their simplicity and no degraded performance, we have used GRU cells for our RNN, as discussed in Chapter 3.

The mathematical implementation of GRU cell is given in Eq. 2.6. x_t and h_{t-1} are the current inputs and previous cell state. The update and reset gates are represented by z_t and r_t respectively. The GRU cell has weights W, U, W_r, U_r, W_z and U_z. Please note that like the neurons in Sec. 2.1.1, the neurons in GRU cell also have bias terms, which we have ignored in here to make equations comprehensible. However, the bias terms are considered during the implementation.

$$\begin{aligned}
h_t &= z_t h_{t-1} + (1 - z_t)\tilde{h}_t \\
\tilde{h}_t &= \tanh\Big(W x_t + U(r_t \odot h_{t-1})\Big) \\
r_t &= \sigma\Big(W_r x_t + U_r h_{t-1}\Big) \\
z_t &= \sigma\Big(W_z x_t + U_z h_{t-1}\Big)
\end{aligned} \tag{2.6}$$

Chapter 3

Methodology

3.1 Overview

In this chapter, we talk about our model that we developed to handle the task of future trajectory prediction in dynamic scene. Briefly, our model takes the past trajectory and the scene context (image of the scene) as inputs to predict the future trajectory of the agent (vehicle/pedestrian/cyclist etc) under consideration. Our model consists of following main components – (1) RNN-ED (RNN Encoder Decoder); (2) SCF (Scene context fusion); and (3) AM (Attention mechanism). Let us first briefly talk about these components to get an intuition of the underlying working of the model, and then discuss each of them in detail in the remaining part of the chapter.

The first component, RNN-ED, is used for the time series prediction. The basic idea behind Encoder-Decoder to use RNN (Recurrent Neural Network, explained in Chapter 2) is to encode the given input series into a latent space and then use the encoded representation to decode, using another RNN, into the output series. RNN-ED has been widely used in sequence prediction tasks such as language translation [8, 18, 54], behavior prediction [7, 31, 38], time series prediction [31, 38, 46].

As mentioned earlier, the inputs to our model are past trajectory and scene (image). Intuitively, the role of SCF is to fuse the scene information and interactions with the trajectory information. To get the scene information, we employ CNN. Briefly, a CNN is used to map the input image into a feature space. Similarly, the input trajectory is mapped to another feature space of velocity features. And the interactions are obtained by mapping the velocity features of neighbouring agents to another feature space representing the concentration of nearby agents. Finally these feature vectors are concatenated. The hope is that this common feature space will have more information than any one of them individually. The SCF is motivated by the recent work [38].

The last component, Attention Mechanism (AM), computes a weighted sum of all the past summaries wrt the decoder's hidden state. The model learns to *attend* more to that summary in the past that relates the most to the situation in future. This can be interpreted as that the agent would exhibit a motion pattern similar to what it did in a past situation that resembled the most to the future scenario. The use of AM is motivated by the use of attention in [8, 46].

One of the major insights that we use in the following is that velocity is a representation of agent motion characteristics, as presented in [7, 38]. During driving, we only look at the motion of the agents nearby and underrstand its motion characteristics. We build on this intuition to consider velocity as a representation of motion context.

In the following, we first present the inputs, outputs and the training objective of our model.

Next, we have detailed the model architecture and its several modules namely CNN, RNN-Encoder, RNN-Decoder, Scene Context Fusion (SCF) and Attention-Mechanism (AM).

3.2 Inputs, Outputs and Training Objective

Let the scene under consideration be represented by image I_0 with size $im_H \times im_W \times 3$. For consistency among the examples, we scale I_0 into image I of size $H \times W \times 3$ using bilinear-interpolation. I is considered as the image of the scene and used as input to the CNN as shown in Fig. 3.1.

We consider N number of agents moving in the scene, where N varies from scene to scene. The ground truth for the past and future positions are represented by $\mathcal{X}$ and $\mathcal{Y}$ respectively, where $\mathcal{X} = [X_1, X_2, \cdots, X_N]$ and $\mathcal{Y} = [Y_1, Y_2, \cdots, Y_N]$. With reference to the current time instant 't', the past and the future trajectories of i^{th} agent are represented by $X_i = [x_{i,t-\nu+1}, \cdots, x_{i,t}]$ and $Y_i = [y_{i,t+1}, y_{i,t+2}, \cdots, y_{i,t+\delta}]$, respectively. Please note that ν and δ represent the number of past and future time steps considered in the model. We have focused on traffic scenario, hence each $x_{i,t}$ and $y_{i,t}$ is a vector in $\mathbb{R}^2$. Similarly the i^{th} agent's past velocity is represented by $\dot{X}_i = [\dot{x}_{i,t-\nu+1}, \cdots, \dot{x}_{i,t}]$. We compute the velocity through temporal convolution layer over 3 successive positions with fixed parameters. We use $\dot{\mathcal{X}} = [\dot{X}_1, X_2, \cdots, \dot{X}_N]$, instead of $\mathcal{X}$, as one of the input to our model (refer to Fig. 3.1).

The objective is to learn the mapping function that allows us to predict future positions represented by $\hat{\mathcal{Y}}$. The predicted future position should be as close as possible to the ground truth of the future positions i.e. the error between $\hat{\mathcal{Y}}$ and $\mathcal{Y}$ should be minimum. Hence, we use L_1 loss function, averaged over the training sequences, as given in Eq. 3.1. It directly measures the absolute error between the prediction $\hat{\mathcal{Y}}$ and the ground truth $\mathcal{Y}$. Thus, the objective is to learn the parameters of the model such that the regularized loss is minimized. We use back-propagation [48] algorithm to optimize our model so that the mean absolute error is reduced.

$$L = \frac{1}{N} \sum_{i=1}^{i=N} ||Y_i - \hat{Y}_i||_1 \tag{3.1}$$

3.3 Model Architecture

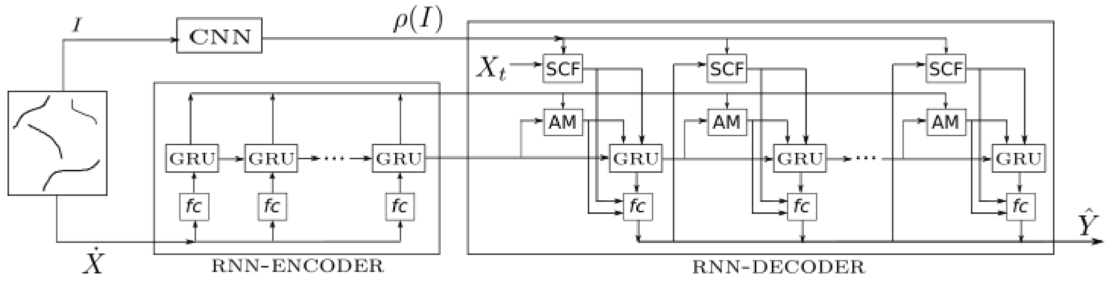


Figure 3.1: Model architecture

We now talk about all the components of the architecture of our model as shown in Fig. 3.1. Briefly, we use the image I and get its feature through the CNN pipeline (shown in detail in Fig. 3.3). We feed the past velocity $\dot{\mathcal{X}}$ into RNN-Encoder to get the summary of motion at all past time steps. All these summaries are then fed into RNN-Decoder. At each future time step, the decoder *attends to* them wrt its state through the 'AM' module and then uses their weighted sum. We obtain the scene context features (agent motion context, scene and interactions) through the 'SCF' module. The decoder cell takes the weighted summary, the scene context and previous cell state as inputs and then predicts the position $\hat{Y}_{i,t}$ at time step 't'. $\hat{Y}_i$ is compared to Y_i to get the loss using Eq. 3.1. This loss is back-propagated [48] to get the gradients at every neuron in the model and accordingly tune the model parameters. Let us now understand all these components in detail.

Please note that, in order to avoid clutter in equations, we omit i wherever necessary. For example, in situations, where we do not talk about interactions, X_t implies $X_{i,t}$, which is the position of the i-th agent under consideration at t-th time step. Similarly for other variables.

3.3.1 RNN-Encoder

We develop the Encoder using GRU (see Chapter 2) based RNN as in [18]. We prefer GRU over LSTM [29] for its simplicity and equivalent performance [19]. The input to the encoder is the past velocity $\dot{X}_t$ and the output is a vector h_t in the latent embedding space. Intuitively, h_t represents the summary of the past trajectory at t-th time step.

As stated in Sec. 3.1, velocity is a dense representation of agent motion context. Due to its compactness, if we use it directly in the encoder, we may lose some characteristics of agent's motion, which may be significant later on. It would, thus, be better to use the high-dimensional representation of velocity as inputs. Hence, we use velocity's high-dimensional representation F_t obtained using Eq. 3.2 as input to the GRU cell. This mapping is represented by the *fully-connected (fc)* block in Fig. 3.1.

$$F_t = \max(0, w_v \dot{X}_t + b_v) \tag{3.2}$$

The GRU cell takes F_t and previous cell-state h_{t-1} as inputs. Similar to as described in Chapter 2, the cell state is updated to h_t using Eq. 3.3. The reset and the update gates, represented by r_t and z_t, decide on how much of the encoder state should be updated or reset subject to the current inputs and the previous state.

$$\begin{aligned}
h_t &= z_t h_{t-1} + (1 - z_t)\tilde{h}_t \\
\tilde{h}_t &= \tanh\Big(W F_t + U(r_t \odot h_{t-1})\Big) \\
r_t &= \sigma\Big(W_r F_t + U_r h_{t-1}\Big) \\
z_t &= \sigma\Big(W_z F_t + U_z h_{t-1}\Big)
\end{aligned} \tag{3.3}$$

Parameters and initialization We initialize the encoder with zeros since we do not have any information about the agent's motion prior to this time instant. From Eqs. 3.2 and 3.3, it can be seen that the parameters to be learnt in the encoder are w_v, b_v, W, U, W_r, U_r, W_z and U_z. Please note that we have intentionally omitted the bias terms in Eq. 3.3 for easy comprehension. They are however considered in the code.

3.3.2 RNN-Decoder

Similar to the encoder, we use GRU based RNN as the decoder. Let us consider the cell at the future time step 't'. The decoder's task is to predict $\hat{Y}_t$ conditioned on the past summary and the scene context features at 't'. Thus, the decoder's GRU cell takes as inputs (refer to Fig 3.1)– (i) the scene context fusion feature $F_t^{'}$ (combination of agent motion context, scene feature, and interaction feature); (ii) weighted past summary c_t obtained using attention mechanism (AM); and (ii) previous cell state $h_{t-1}^{'}$. The scene context features $F_t^{'}$ given by Eq. 3.4 comprises of agent motion context $f_{\dot{\hat{Y}}_{t-1}}^{'}$, image features $scf_{t-1}^{'}$ and interactions among agents $aif_{t-1}^{'}$. The details of of scene context fusion feature and weighted summary determination (AM) is given in Secs. 3.3.3 and 3.3.4, respectively.

$$F_t^{'} = \left[f^{'}_{\dot{\hat{Y}}_{t-1}}, scf^{'}_{t-1}, aif^{'}_{t-1}\right]^T \tag{3.4}$$

We update the GRU cell to state $h_t^{'}$ using Eq. 3.5. $r_t^{'}$ and $z_t^{'}$ are the reset and update gates for the decoder that decide on how much to reset/ update the cell state $h_{t-1}^{'}$ conditioned on the summary c_t and the inputs $F_t^{'}$.

$$\begin{aligned}
h_t^{'} &= z_t^{'}h_{t-1}^{'} + (1 - z_t^{'})\tilde{h_t^{'}} \\
\tilde{h_t^{'}} &= \tanh\Big(W^{'}F_t^{'} + U^{'}(r_t^{'} \odot h_{t-1}^{'}) + V^{'}c_t\Big) \\
z_t^{'} &= \sigma\Big(W_z^{'}F_t^{'} + U_z^{'}h_{t-1}^{'} + V_z^{'}c_t\Big) \\
r_t^{'} &= \sigma\Big(W_r^{'}F_t^{'} + U_r^{'}h_{t-1}^{'} + V_r^{'}c_t\Big)
\end{aligned} \tag{3.5}$$

Note that, $h_t^{'}$ can be interpreted as high dimensional motion characteristics at future time instant 't'. Instead of directly mapping from $h_t^{'}$ to velocity [38], we again condition it with summary c_t (obtained using AM) and the input features $F_t^{'}$ to get the agent motion features $g^{'}_{\dot{\hat{Y}}_t}$ through Eq. 3.6. Similar approach has been followed in [18, 54, 46]. This can be interpreted as recovering relevant information in the inputs and the summary that may have been lost when obtaining the decoder state $h_t^{'}$ through Eq. 3.5.

$$g^{'}_{\dot{\hat{Y}}_t} = W_g^{'}F_t^{'} + U_g^{'}h_t^{'} + V_g^{'}c_t \tag{3.6}$$

Now that we have the agent motion features $g^{'}_{\dot{\hat{Y}}_t}$, we map it back to get the velocity at t-th time. We implement this using a *fully-connected* layer given by Eq. 3.7.

$$\dot{\hat{Y}}_t = w_v^{'}g^{'}_{\dot{\hat{Y}}_t} + b_v^{'} \tag{3.7}$$

The predicted position $\hat{Y}_t$ is next obtained using predicted velocity $\dot{\hat{Y}}_t$ and previous predicted position $\hat{Y}_{t-1}$ through Eq. 3.8. We also know that the agents under consideration would lie within the image boundaries. We, thus, use this information to limit the predicted position $\hat{Y}_t$ to lie within the image boundary.

$$\hat{Y}_t = \hat{Y}_{t-1} + \dot{\hat{Y}}_t \tag{3.8}$$

Parameters and initialization The Encoder-Decoder represents a continuous sequence of GRU cells that aim to extract the motion characteristics at each time step. The output of the encoder's 1^{st} cell is used as input to the encoder's 2^{nd} cell and so on. Similarly the output of decoder's 1^{st} cell is used as input to the deocder's 2^{nd} cell. Following this relationship, we initialize the decoder's first cell with the encoder's last cell state, as shown in Fig. 3.1. This process is similar to that used in [18, 54].

It is important to note that as in Eq. 3.3, we have here as well skipped the bias terms in Eqs. 3.5 and 3.6 for easy understanding. And the parameters to be learnt in the decoder are $W^{'}$, $U^{'}$, $V^{'}$, $W_z^{'}$, $U_z^{'}$, $V_z^{'}$, $W_r^{'}$, $U_r^{'}$, $V_r^{'}$, $W_g^{'}$, $U_g^{'}$, $V_g^{'}$, $w_v^{'}$ and $b_v^{'}$.

3.3.3 Scene Context Fusion (SCF) and Interactions

As discussed in Sec. 3.1, we humans analyze the scene context and understand the relationship between the agent motion context, scene and interactions. We use this understanding to predict the future position of the agents. Inspired by this, we try to mimic the same process using *Scene*

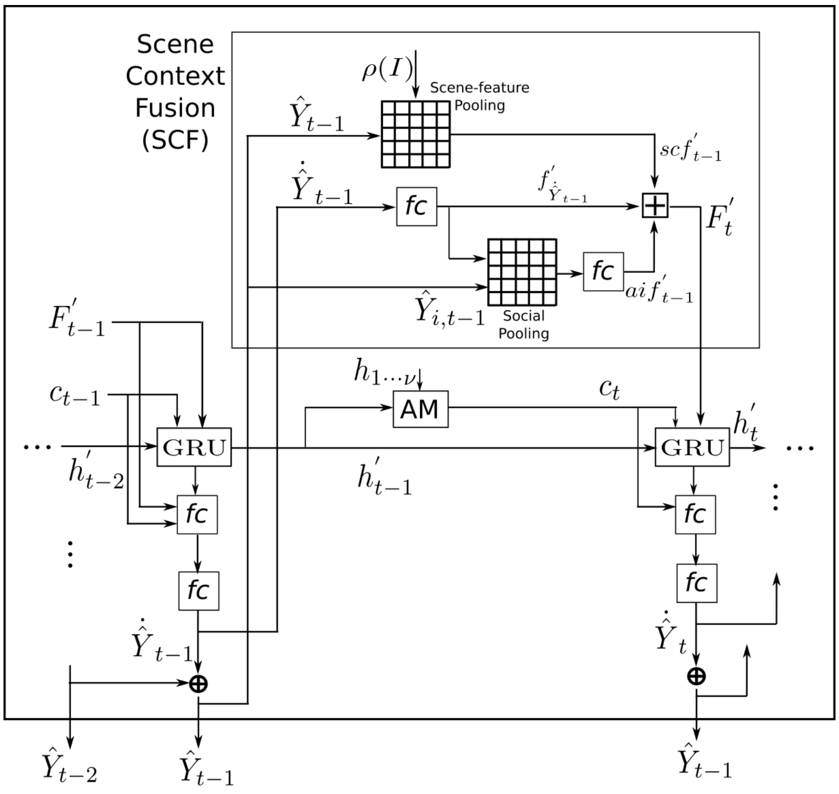


Figure 3.2: Scene Context Fusion

Context Fusion (SCF) module as shown in detail in Fig. 3.2. It fuses the agent motion context, image features and interactions among the agents into a common feature space. Its a compact representation of the agent's motion subject to its interactions with the scene and nearby agents. Without loss of generality, let us focus on agent 'i' at time step 't'. The SCF takes as inputs – (i) the predicted position $\hat{Y}_{i,t-1}$ & velocity $\dot{\hat{Y}}_{i,t-1}$ of the i^{th} agent at previous time step; (ii) image feature $\rho(I)$ obtained using CNN; and (iii) position $\hat{Y}_{j\setminus i,t-1}$ & velocity $\dot{\hat{Y}}_{j\setminus i,t-1}$ of all other agents in the scene at previous time step.

An important point to note here is that these features vary with time. Hence, we are able to model the dynamic scene environment. Another major aspect of the SCF is that it uses predicted position and velocity instead of using their ground truth values. We have done this due to the following intuitions:

- The usage of predicted values instead of ground truth would provide the network with higher gradients that would help improve the learning process and thereby yield better performance.
- If we use the ground truth values, perhaps the network would converge quickly due to availability of correct information from the beginning itself. But a drawback of this would be that it would never understand how an error in those parameters may propagate and corrupt the results. During test performance evaluation, the ground truth will not be available for these parameters. Hence, the network would have to depend on their predicted values only. And since it would not be trained that way, the control in the propagation of error with time will be difficult. As a result, its test performance would degrade.

Thus, it is evident that the SCF takes into account the dynamic scene features and would also yield a higher test performance. The different aspects of SCF viz. agent motion context, scene features and interactions are presented next.

Agent Motion Context

As done in Sec. 3.3.1, we here as well use velocity features as representation of agent's motion. Thus, predicted velocity $\dot{\hat{Y}}_{t-1}$ is mapped to a high-dimensional representation $f'_{\dot{\hat{Y}}_{t-1}}$ using Eq. 3.9. This mapping is represented by the '*fc*' block after $\dot{\hat{Y}}_{t-1}$ in Fig. 3.2. The relationship between the velocity and motion context should be consistent throughout the model. Hence, we have used the same weights and biases (w_v and b_v from Eq. 3.2) in Eq. 3.9.

$$f'_{\dot{\hat{Y}}_{t-1}} = max(0, w_v \dot{\hat{Y}}_{t-1} + b_v) \tag{3.9}$$

Scene Features and Pooling

To understand the effect of scene on the agent's motion, we need to identify the image features, such as presence of roads, crosswalks, buildings etc. And we know that CNNs are powerful enough to identify such features. So, we first obtain the image features using CNN pipeline as given in Fig. 3.3. The resized image I is passed through a series of convolution layers to yield $\rho(I)$ which represents image features and has dimensions of $H_{CNN} \times W_{CNN} \times D_{CNN}$. These are used subject to the position of the agent for scene-feature pooling, explained below. The CNN is built with 2 layers. The weights and biases for these layers are represented by (w_{k_1}, b_{k_1}) and (w_{k_2}, b_{k_2}) respectively. We use $ReLU$ activation function (see Chapter 2) for both the layers.

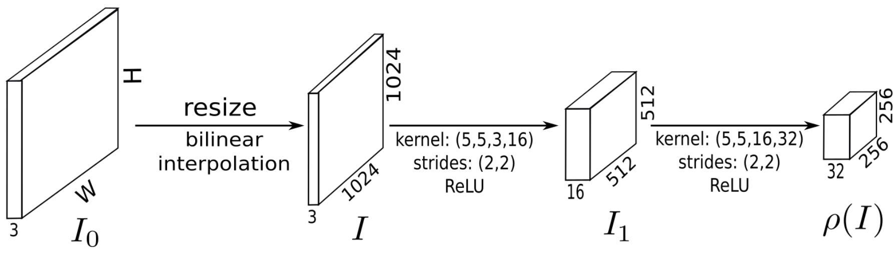


Figure 3.3: CNN for image features determination

The features in I_0 would be contained in $\rho(I)$, the output of our CNN pipeline. Its, thus, important to note that we do not explicitly provide the image features to our model, rather it learns them on its own. However, I_0 has size $im_H \times im_W \times 3$, whereas $\rho(I)$ has size $H_{CNN} \times W_{CNN} \times D_{CNN}$. This means that the features at position $\hat{Y}_{t-1}$ in I_0 would correspond to the features at its corresponding scaled position $\hat{Y}_{t-1,scaled}$ in $\rho(I)$. With reference to the size of image features and the image, the scale factor is intuitively obtained through Eq. 3.10, where $\odot$ represents element-wise multiplication. Using this philosophy, the image features at $\hat{Y}_{t-1}$ are obtained by pooling features from $\rho(I)$ at $\hat{Y}_{t-1,scaled}$. We represent these pooled features by scf'_{t-1}. And since we use the predicted position while pooling, we get scene features that change with motion of the agent. This pooling operation is pictorially represented by '*Scene Feature Pooling*' block in Fig. 3.2.

$$\hat{Y}_{t-1,scaled} = \hat{Y}_{t-1} \odot \left[\frac{W_{CNN}}{im_W}, \frac{H_{CNN}}{im_H}\right] \tag{3.10}$$

Interactions among agents

From Sec. 3.1, we know that the interactions among the agents play an important role in their motion characteristics and also that velocity compactly represents the motion context. Hence, we build on this intuition and use the velocity features to get the interaction features.

Let us focus on the motion of agent 'i'. It is intuitive to say that its motion would be impacted more by those agents '$j \setminus i$' that are closer to it. Its motion may also be significantly impacted by a far-off agent if that has say high relative velocity. So it can be seen that we could get the interaction features by using the velocity features and the positions of the agents.

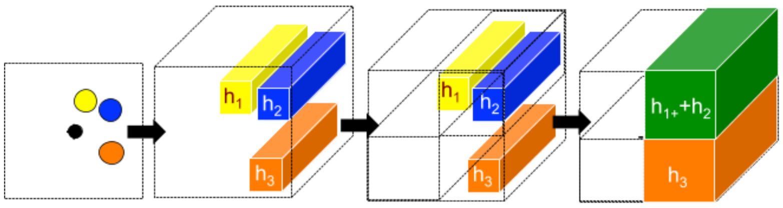


Figure 3.4: Interaction features pooling using *Social-Pooling*. Taken from [7]. Black dot signifies the position of i^{th} agent. The velocity features of neighboring agents are placed at their corresponding positions in the $N_0 \times N_0$ grid. h_1, h_2 and h_3 represent the velocity features of agents 1, 2 and 3 respectively.

We use the *Social-Pooling* concept introduced in [7] to get the interaction features. An agent's motion would be impacted only by those agents that lie within some distance. Hence, we focus on the velocity features of only those agents that lie in some neighborhood around the agent under consideration, as aptly shown in Fig. 3.4. Usually I_0 would have a large size. Due to this, the agents $j \setminus i$ impacting the motion of i^{th} agent maybe situated at quite a distance from i. To include them for the interaction feature pooling, we would have to construct a large grid, which would be computationally heavy. So, we first scale down I_0 to size $N_1 \times N_1$ and place the velocity features of all the agents at their corresponding scaled down positions in this $N_1 \times N_1$ grid. We now consider a region $N_0 \times N_0$ around the agent 'i' as shown in Fig. 3.4. This region signifies that the agents beyond it do not impact the motion of agent 'i'. We hence, pool the velocity features of all those agents that lie in this region. This pooling, represented by H^i_{t-1} is done through Eq. 3.11 and represented by '*Social Pooling*' block in Fig. 3.2. Due to the scaling, it can be seen from Fig. 3.4 that the velocity features are summed up for the agents that lie in the same grid location. This can be interpreted as a higher concentration of interaction features at such locations in the grid.

$$\begin{aligned} H^i_{t-1}(m,n,:) &= \sum_{j\in N} 1_{mn}\left[x^{j\setminus i}_{t-1} - x^i_{t-1}, y^{j\setminus i}_{t-1} - y^i_{t-1}\right] f^{'}_{\dot{Y}_{j\setminus i,t-1}} \\ \hat{Y}_{i,t-1} &= [x^i_{t-1}, y^i_{t-1}] \\ \hat{Y}_{j\setminus i,t-1} &= [x^{j\setminus i}_{t-1}, y^{j\setminus i}_{t-1}] \end{aligned} \tag{3.11}$$

In Eq. 3.11, 1_{mn} is an indicator function about the presence of j at (m, n) position in the $N_0 \times N_0$ grid around $\hat{Y}_{i,t-1}$. It is thus, equal to 1 if j is present, else it is 0. We next obtain a dense representation of interaction features by mapping $H^i_{t-1}(m,n,:)$ to $aif^{'}_{t-1}$ using a *fully-connected* layer as given by Eq. 3.12 and aptly shown by the '*fc*' block after '*Social Pooling*'.

$$aif^{'}_{t-1} = w_i H^i_{t-1} + b_i \tag{3.12}$$

Following are the important aspects of the way we have implemented interaction feature pooling:

- Unlike [7] which uses hidden states for interaction feature pooling, we have used velocity features. We have done this with regards to the following reasons:
  - Due to the *attention* mechanism, we condition the hidden states $h^{'}_t$ with the summary c_t and the previous states $h^{'}_{t-1}$ and map it to $g^{'}_{\dot{Y}_t}$ to get the motion context features,

as discussed in Sec. 3.3.2. Thus, for us, its $g^{'}_{\dot{Y}_t}$ that actually represents the velocity features instead of $h^{'}_{t-1}$.

  - We have constrained the predicted position to lie within the image boundary as discussed in Sec. 3.3.2. Due to this, if the position is truncated, it would mean a change in predicted velocity too. Hence, it should be better to accordingly correct the earlier steps as well. But it would then make the model too complicated. Further, we also know that the agent motion is better represented by the velocity features. Thus, moving ahead with our intuition and keeping the model simple, we have used the velocity features for pooling of interaction features.

- Since, we consider all the agents in the scene, we are able to model the interactions due to change in positions of all agents simultaneously. Hence, the interaction feature pooling is completely dynamic. Due to this, agents that may not be interacting at some time may become interactive at some later time in the future.

- A similar implementation of *Social-Pooling* has been carried out in [38]. A cropped region around the agent i is taken under consideration. It is assumed that no other agent outside this region affects the motion of any of the agents lying inside it. But there could be situations wherein an agent $k \setminus i$ lying right outside the region would be impacting some agent $j \setminus (i, k)$ inside the region. Situation could be worse when there would be a high concentration of such agents. Hence, such agents k would be directly impacting j and thereby indirectly impacting i. Thus, we can see that the usage of cropped region from the beginning itself is not good. We have overcome this problem by looking at the entire image scene and then pooling wrt dynamic movements of all the agents. Hence, our model is able to capture those indirect relationships as well.

From the above salient points, it is evident that our model closely resembles a real-life dynamically changing traffic scenario. *Parameters* to be learnt in the SCF module are w_i and b_i.

3.3.4 Attention Mechanism (AM)

When we humans drive, we expect that in future, the agents would exhibit a behavior similar to what they did in a similar past situation. For example, if an agent attempted to change lanes in a busy traffic scenario in the past, we expect them to do the same in future. Hence, when we predict the future motion of agents, we intelligently recall their behavior from a similar past. This makes sense since for the short duration of past and future that we consider, we don't expect the motion pattern to be very different in similar situations. Thus, while driving, we *attend* more to similar past situations and less to the others.

Similar concept has been implemented in [8, 46] for neural language translations and finance stock predictions. We have taken inspiration from these works and implemented an *attention-based summary* for our case as against the naive usage of constant summary in [38] for a traffic scenario model similar to ours. We call it as *Attention-Mechanism* and have incorporated it into our model through the '*AM*' block in Fig. 3.1. Its detailed architecture is presented in Fig. 3.5.

We know that the hidden states of the encoder represent the summary of motion at all time steps in the past. We use this information to build upon the above discussed concept. If we compare all the states of the encoder to the current decoder state, we could find out which past situation resembles the most to the future situation. Accordingly, we could weigh them all and use their weighted sum as the summary. We achieve this through Eq. 3.13 to get the weighted summary c_t. It can be seen in Eq. 3.13 that hidden states (at all time steps in the past) h_m are used with the decoder state h'_{t-1} to get their corresponding weights $\alpha_{t,m}$. These weights $\alpha_{t,m}$ are then used with h_m to get the weighted summary c_t. Please note that ν in Eq. 3.13 is the number of past time steps as mentioned in Sec. 3.2.

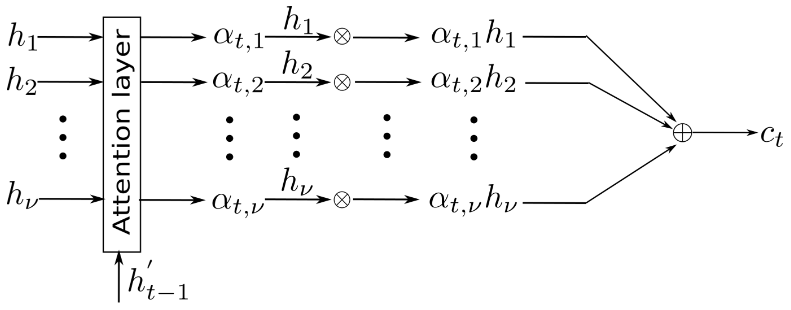


Figure 3.5: Attention Mechanism

$$\begin{aligned} c_t &= \sum_{m=1}^{m=\nu} \alpha_{t,m} h_m \\ \alpha_{t,m} &= \frac{e_{t,m}}{\sum_{m=1}^{m=\nu} e_{t,m}} \\ e_{t,m} &= V_a^T \tanh\left(U'_a h'_{t-1} + W_a h_m\right) \cdots \forall m \in (1,\nu) \end{aligned} \tag{3.13}$$

Parameters With reference to Eq. 3.13, the parameters to be learnt in 'Attention-Mechanism' are U'_a, W_a and V_a. We have intentionally skipped the bias terms in Eq. 3.13 for easy understanding. However, we have considered them in the code.

Chapter 4

Experiments and Results

In this chapter, we first present the experimental setup viz. the dataset used, dimensions/ values of the model parameters, evaluation metrics, baselines for comparison and learning criteria. Later on we present the results and the observations. In the end, we also perform some of the ablation studies that we carried out in order to better understand our final model.

4.1 Experimental Setup

4.1.1 Dataset

Since our focus is to understand behavior of agents in traffic situation, we use *Stanford Drone Dataset (SDD)* [49] for our experimental and model evaluation purposes. SDD is a highly challenging real-world dataset that has been used in many state-of-the-art methods such as the most recent one called DESIRE [38]. Please note that though we have trained our model on a traffic scenario dataset, it could be extended to model 3-D motion scenario (for aerial agents) or some other time prediction series tasks.

SDD is a collection of videos captured from aerial drones at several locations of the Stanford university campus and named – bookstore, coupa, deathCircle, gates, hyang, little, nexus, quad – with 7, 4, 5, 9, 15, 4, 12 and 4 videos at these locations respectively. The videos have several agents (bicyclist, pedestrian, skateboarder, cart, car and bus) moving in highly dynamic situations (cross-roads/roundabouts etc.) with variable motion characteristics (slow/fast moving, sharp maneuver, static, etc.). The SDD also provides an annotation file for each of the videos. The annotation file contains tracking ids of the agents, their corresponding bounding box co-ordinates and labels of lost, occluded or generated across the time frames. The videos are recorded at 30 fps and vary in lengths from around 1 to 10 minutes.

Preprocessing

We considered the past and future motions for 2 and 4 seconds respectively at 10 fps, as done in [38]. So, we extract the images (or frames) of the scene across the videos in SDD by preprocessing it as follows:

- We ignored the unstable camera videos (deathCircle: video3; nexus: video3, video4, video5), the lost labels data (bookstore: video1) and hyang: video3 (due to problem in extraction of frames). Thus, we have a total of 54 videos left with us.
- We determine the position of all the agents by considering it at the centre of the bounding box.

- We build examples by (i) extracting the frames at 10 fps and (ii) determining the current, past (2 seconds) and future (4 seconds) positions. Please note that there are examples, where an agent is lying very close to the image boundaries. For such cases, either the agent would be entering into the frame or exiting it. In those situations, the past or future may not be available for total of 2 or 4 seconds respectively. We handle these cases by zero padding for the time steps, where agent is not present.

To give you an insight of the complexity of the SDD regarding agents' motion subject to the scene and interactions, we present two such examples extracted from SDD in Fig. 4.1. Several agents are seen to be moving subject to the scene (roads, roundabout etc) such that they avoid collisions with each other.

Please note that we use the pixel location as the position, since position is not explicitly provided in standard terms of meters.

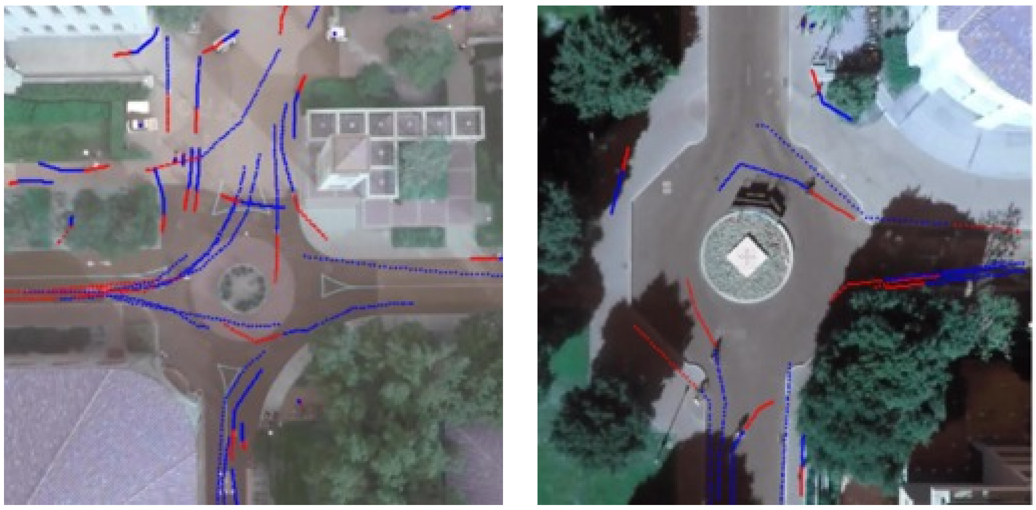


Figure 4.1: Examples extracted from SDD [49] after preprocessing. Red and blue depict past and future motions respectively. Agents can be seen to be moving subject to the scene (road, roundabout, side-walks etc) and ensuring no collisions with each other.

Following the above preprocessing steps, we extracted around 15000 frames at 10 fps and considered them as our examples. Each frame contains the image of the frame and past (2 seconds), current and future (4 seconds) positions of the agents.

Data split

We next randomly split the set of 54 videos into 5 folds, and randomly select one of the folds as our test set and the remaining as train set. Please note that we did not carry out 5 fold cross-validation. Finally, we obtained around 11500 and 3500 examples (frames) for train and test datasets, respectively.

4.1.2 Code

We develop the entire code base using the well known TensorFlow [6] library. Note that because of the unavailability of the code for DESIRE [38], we had to develop the baseline codes ourselves. Since, we did not carry out 5-fold cross validation and that the data split is also not the same as used in DESIRE; hence our results presented in Table 4.2 cannot be directly compared to that quoted in DESIRE.

We will make our code publicly available in due course of time, when we are done with further work/experiments, as presented in Sec. 4.3 and Chapter 5.

4.1.3 Model Parameters

As we consider past and future for 2 and 4 seconds respectively at 10 fps (Sec. 4.1.1), thus, the number of GRU cells in RNN-Encoder and RNN-Decoder are 20 and 40 respectively. Our model handles the case of past or future less than 2 and 4 seconds as follows:

- Only those hidden states of RNN-Encoder are considered in attention-mechanism, where agent is present. This is necessary since it would be wrong to consider hidden states where agent is absent.
- During training, we consider only those time steps in future for loss (Eq. 3.1) determination where agent is present.

We have implemented the RNN-Encoder and RNN-Decoder with 48-dimensional cell state size and single-layered GRUs. We pool the scene features (Sec. 3.3.3) and interaction features (Sec. 3.3.3) over 1×1 and 5×5 regions respectively. The dimensions/ values of the parameters that we have used in the model are given in Table 4.1. As mentioned in Sec. 3.3, we would like to reiterate about the parameter sharing as follows:

- w_v, b_v that are used to map from velocity to velocity features are shared in RNN-Encoder and SCF at all time steps.
- The GRUs of the RNN-Encoder share the parameters (W, U, W_r, U_r, W_z and U_z).
- The GRUs of the RNN-Decoder share the parameters ($W^{'}$, $U^{'}$, $V^{'}$, $W^{'}_z$, $U^{'}_z$, $V^{'}_z$, $W^{'}_r$, $U^{'}_r$, $V^{'}_r$, $W^{'}_g$, $U^{'}_g$, $V^{'}_g$, $w^{'}_v$ and $b^{'}_v$).
- The parameters w_i and b_i in interaction feature pooling are shared at all time steps in RNN-Decoder.
- The Attention-Mechanism (AM) module shares the parameters $U^{'}_a$, W_a and V_a at all time steps in the RNN-Decoder.

4.1.4 Evaluation Metrics

We used L_2 distance error (scaled down by 5) between Y and $\hat{Y}$ at different time steps as test performance evaluation metrics, same as that used in [38]. We preferred L_2 since its normally used across related works and gives the direct distance error between the ground truth and the prediction.

Please note during performance evaluation, we consider only those time steps in future, where agent is present. Since we quote the mean error, hence the lower the error, the better the accuracy. Thus, we expect the mean error to reduce as we increase the complexity from RNN-ED to our final model (RNN-ED-VSI-A).

4.1.5 Baselines for comparison

We developed the following variants of our model and evaluated them for comparison.

- **RNN-ED-DESIRE**: This is an RNN-Encoder-Decoder as implemented in [38] without the Scene Context Fusion module. In this, the decoder's 1^{st} cell is initialized with zeros with its input being the encoder's last cell's hidden state. The inputs to the remaining decoder

Table 4.1: Model Parameters

Model	Parameters	Dimensions/Values
CNN	H, W	1024
	H_{CNN}, W_{CNN}	256
	w_{k_1}	$5 \times 5 \times 3 \times 16$
	w_{k_2}	$5 \times 5 \times 16 \times 32$
	b_{k_1}	16
	b_{k_2}	32
Encoder	w_v	16×2
	b_v	16
	W, W_r, W_z	48×16
	U, U_r, U_z	48×48
Attention Mechanism	$U_a^{'}$, W_a	48×48
	V_a	1×48
SCF	w_i	$16\times$ $(5 \times 5 \times 16)$
	b_i	16
	N_0	5
	N_1	32
Decoder	$W^{'}$, $W_z^{'}$, $W_r^{'}$, $W_g^{'}$	48×64
	$U^{'}$, $U_z^{'}$, $U_r^{'}$, $U_g^{'}$	48×48
	$V^{'}$, $V_z^{'}$, $V_r^{'}$, $V_g^{'}$	48×48
	$w_v^{'}$	2×48
	$b_v^{'}$	2

cells are zeros. This is quite an unusual setup since it does not express the continuity between the Encoder and Decoder as explained in Sec. 3.3.2.

- **RNN-ED**: RNN-Encoder-Decoder variant of our model i.e. our model without the attention mechanism (AM) and scene context fusion (SCF) modules. Inputs to the decoder are considered to be zeros. Since it does not have any attention-mechanism, we map the hidden state $h_t^{'}$ directly to $\dot{\hat{Y}}_t$ instead of going through $g_{\dot{\hat{Y}}_t}^{'}$ as in Eq. 3.6. This is our standard baseline. We progressively add different components (SCF/Interaction and AM) to this baseline and show the effects of each of them.

- **RNN-ED-VSI**: RNN-ED with entire SCF module i.e. $F_t^{'}$ as inputs to decoder. Please note that this model is different from our final model in the only sense that it does not have 'Attention-Mechanism' model. Hence the comparison of this model to our final model would show us the overall impact of the attention-mechanism.

- **RNN-ED-A**: RNN-ED with attention-mechanism. Inputs to decoder are zeros. Hidden state $h_t^{'}$ is used with c_t to map to $g_{\dot{\hat{Y}}_t}^{'}$ and then to $\dot{\hat{Y}}_t$. We use this model to show how the 'Attention-Mechanism' helps our most basic model RNN-ED.

- **Our Model (RNN-ED-VSI-A)**: Our final model that uses RNN-Encoder-Decoder with attention-mechanism (AM) and scene context fusion (SCF) modules, as given in Fig. 3.1.

4.1.6 Learning Criteria

We use Adam Optimizer [34] with a learning rate of 0.001 for training our model. RNNs have problem of vanishing or exploding gradients. To avoid exploding gradients, we clip the gradient with L_2-norm of 1.0. Dataset is split into train and test data ensuring no overlapping videos.

We have implemented our code in TensorFlow 1.0 and experimented on NVIDIA GeForce GTX TITAN Black. Training has been carried out for 30 epochs for all the baselines (Sec. 4.1.5). We used a batch of 32 frames for RNN-ED-DESIRE, RNN-ED and RNN-ED-A. But we could use a batch size of only 4 frames for RNN-ED-VSI and RNN-ED-VSI-A due to the increase in size of inputs in lieu of scene context fusion. We continued with usage of 32 frames for RNN-ED-DESIRE, RNN-ED and RNN-ED-A since higher batch size generally helps in better training and generalization. Thus, if the models with 4 frame batch size perform better than others, it would mean that their performance would be even better if they could be implemented with a higher batch size. It takes around 1 day for training RNN-ED-VSI and RNN-ED-VSI-A and the others are trained in around 3 hours.

4.2 Results

We present a comprehensive summary of the test performance we obtained across all the baselines (Sec. 4.1.5) in Table 4.2. Let us pick the baseline RNN-ED in Table 4.2. The value 1.75 at 1.0 sec means that the mean L_2 error (scaled down by 5) for the test set at 1 seconds is 1.75 pixels.

Table 4.2: Test Error (pixel error scaled down by 5). Different columns represent different prediction horizons.

Baseline	**1.0 sec**	**2.0 sec**	**3.0 sec**	**4.0 sec**
RNN-ED-DESIRE [38]	1.76	3.98	6.51	9.31
RNN-ED	1.75	3.94	6.47	9.26
RNN-ED-VSI	1.78	3.91	6.41	9.22
RNN-ED-A	1.70	3.84	6.32	9.08
RNN-ED-VSI-A (Our Final Model)	**1.70**	**3.79**	**6.22**	**8.92**

We have drawn following important observations from Table 4.2.

- RNN-ED performs better than RNN-ED-DESIRE. This shows that it is better to use encoder's last cell's state to initialize the decoder's first cell than use it as an input.
- RNN-ED-VSI performs marginally better than RNN-ED. This shows that fusion of scene context and interaction is helping to some extent.
- RNN-ED-A's performance is much better than RNN-ED. This shows that the Attention-Mechanism is significantly improving the prediction.
- On comparing our final model (RNN-ED-VSI-A) with other baselines, it is observed that the *attention-mechanism* with the scene context yields much better performance, thereby justifying itself.

In order to better understand the behavior of the algorithms, we visually show the predicted trajectories of the baselines RNN-ED and our final model (RNN-ED-VSI-A) in Figs. 4.2-4.12. In all these examples, it is evident that our model outperforms RNN-ED by capturing the scene context and interactions quite accurately. We have detailed the observations specific to the examples along with the figures.

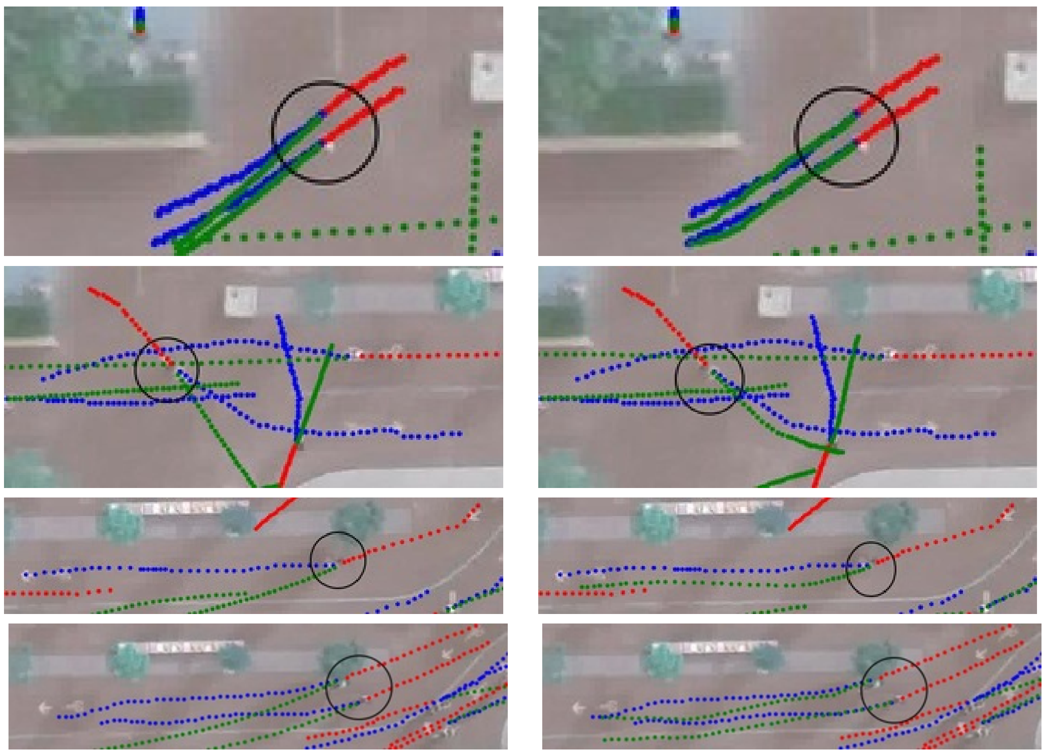
We present analysis of one of the figures to bring forth the effectiveness of our model. Please refer to the fourth row in Fig. 4.2. It can be seen that our model (RNN-ED-VSI-A) is quite accurately able to capture the scene context and interactions among agents. The agent is predicted to be changing its path along the road direction and also ensuring that it does not collide with neighboring agents. This is how agents move in a real scenario. Hence, our model is efficiently able to predict future motion from past motion conditioned on dynamic scene.

4.3 Ablation Studies

Before coming up with our final model, we carried out following experiments:

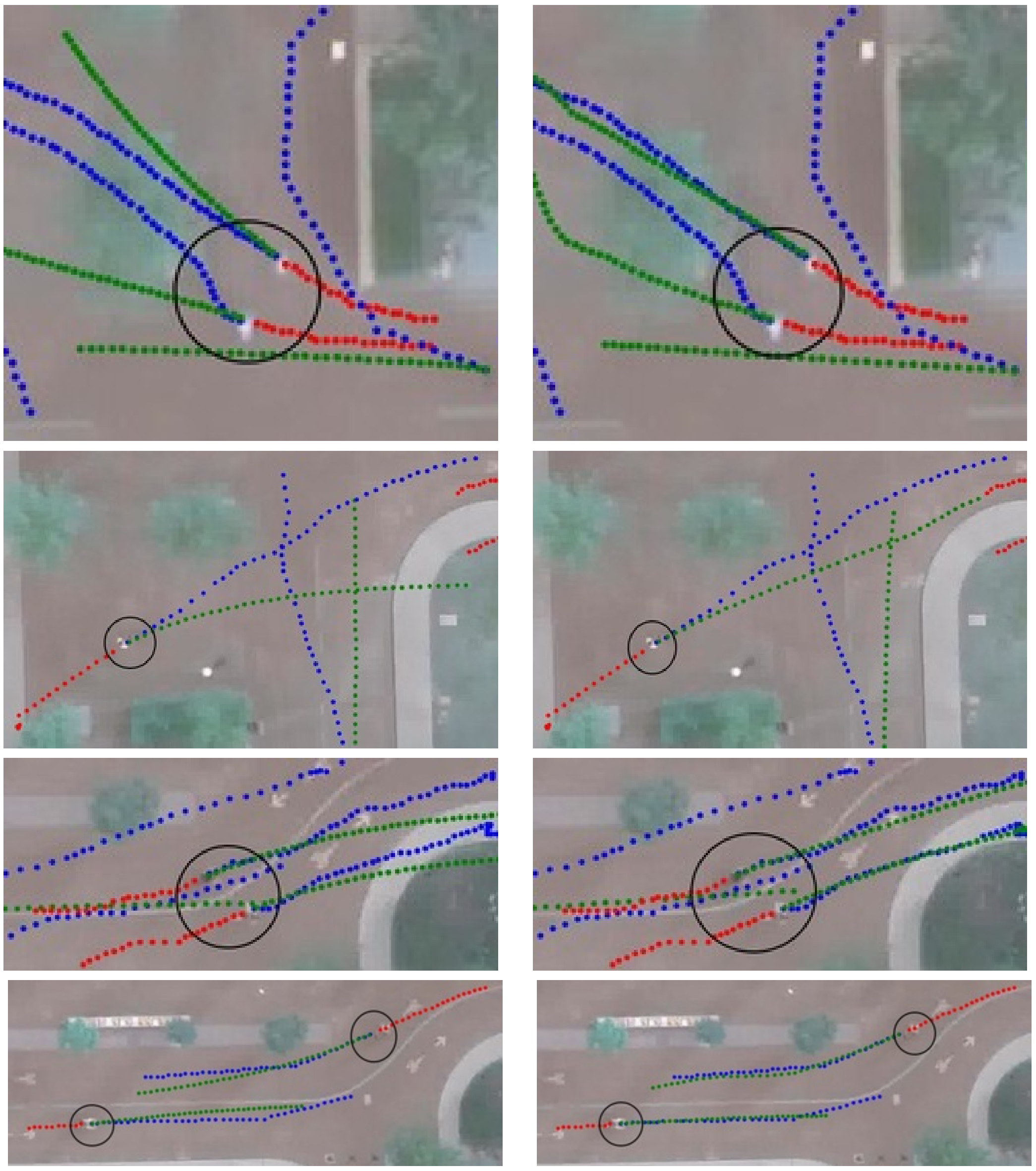
- **Effect of loss functions:** We trained our model with L_2, exponential L_1, and exponential L_2 loss functions, as in [31]. But we found that usage of these losses predicted linear trajectories almost everytime even though the data has many agents that take curved future trajectories. When we changed to L_1 loss-function, we could see curved future trajectories more often, which is much more realistic and also improved the test performance.

- **Fusion of scene features into encoder:** As done in the decoder, we fused scene context and interaction features with velocity features in the encoder as well, but found its test performance to be lower than when using fusion of velocity (predicted) features and image features only in decoder. We thus, fused the scene context and interactions only in the decoder. We are still working in this direction to understand why this happened as it does not make much sense. We intend to address this issue carefully by carrying out further experiments.



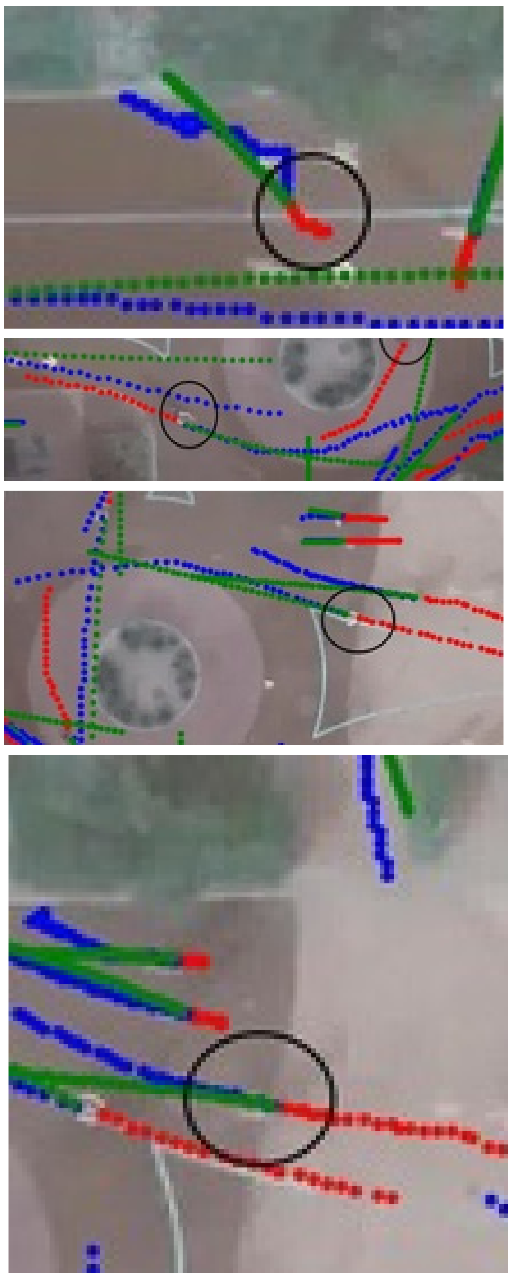
(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.2: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A is able to capture the interactions and predicts such that the two agents under consideration don't collide. **Second row** RNN-ED-VSI-A well captures the scene and predicts the trajectory to be curved along the road. **Third row** RNN-ED-VSI-A captures the scene and interactions. The agent path is predicted to be along the road and also such that it avoids collisions with other agents. **Fourth row** RNN-ED-VSI-A is able to capture both interactions and scene context.



(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.3: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A is able to capture the interactions and predicts such that the agents don't collide. **Second row** RNN-ED-VSI-A is well able to capture the scene context. The predicted path is along the road direction instead of moving into the sidewalk as by RNN-ED. **Third row** RNN-ED-VSI-A very accurately captures the interactions and scene context. The agents are predicted to move with the road direction and ensuring that they don't collide. **Fourth row** RNN-ED-VSI-A is again able to capture the scene context very well. The predicted paths curve along with the roads.


(a) RNN-ED

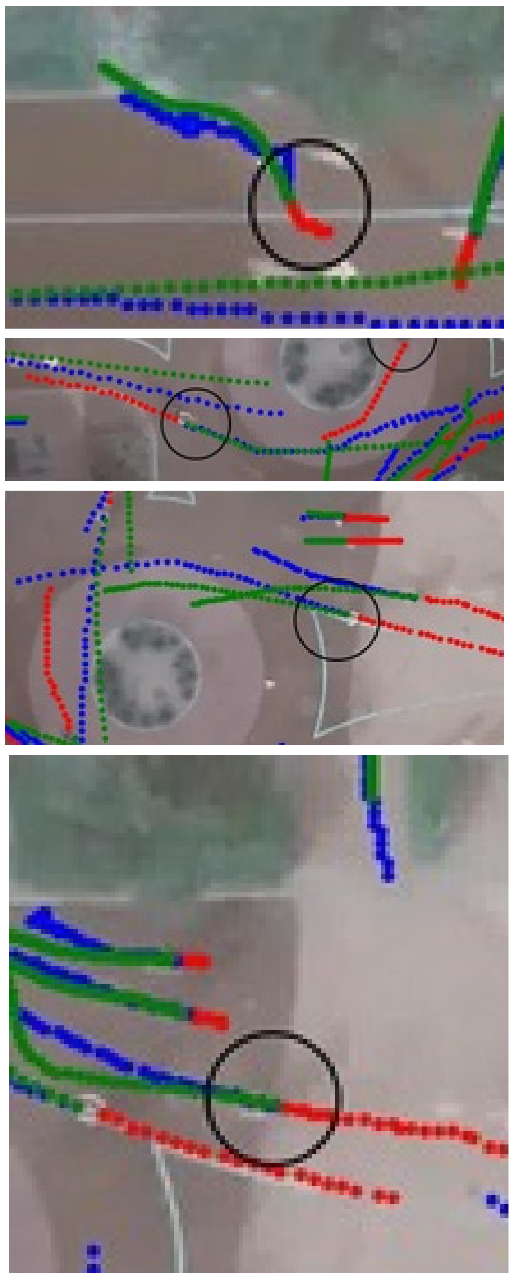
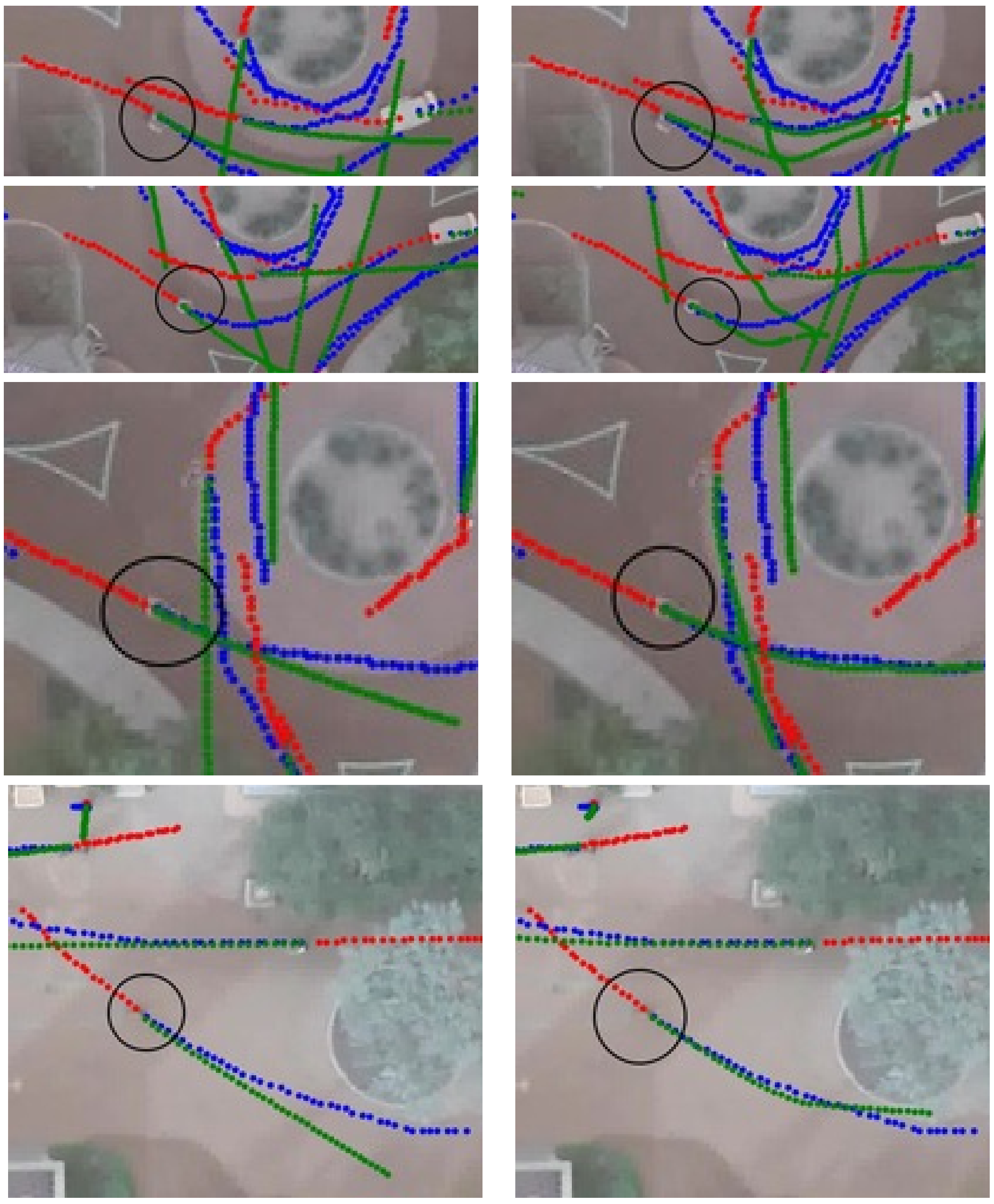
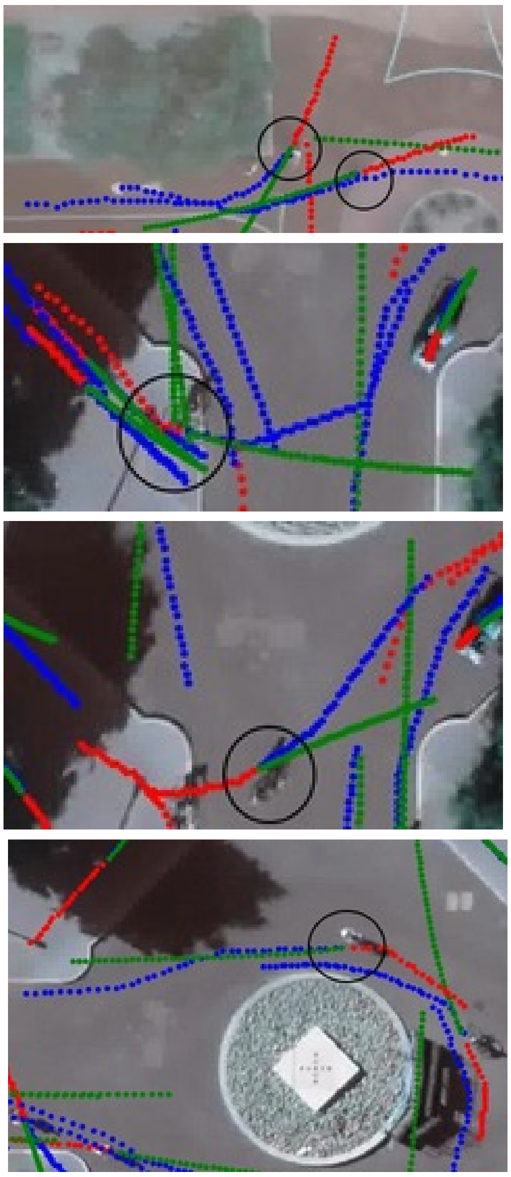

(b) RNN-ED-VSI-A

Figure 4.4: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A is able to capture the sudden changes in curves due to the ‘attention-mechanism’. **Second row** RNN-ED-VSI-A is well able to capture the scene context. The predicted path curves along the circular junction. **Third row** RNN-ED-VSI-A very accurately captures the scene context. The predicted path curves along the circular junction. **Fourth row** RNN-ED-VSI-A is again able to capture the scene context and interactions. The model is able to predict that the agent would move towards its right. The predicted path curves along with the road avoiding collisions as well.



(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.5: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A predicts trajectory that curves with the circular junction. **Second row** RNN-ED-VSI-A is well able to capture the scene context. The predicted path curves along the circular junction. **Third row** RNN-ED-VSI-A very accurately captures the scene context. The predicted path curves along the circular junction. **Fourth row** RNN-ED-VSI-A is again able to capture the scene context and predicts that the agent would turn to follow the driveable area.


(a) RNN-ED

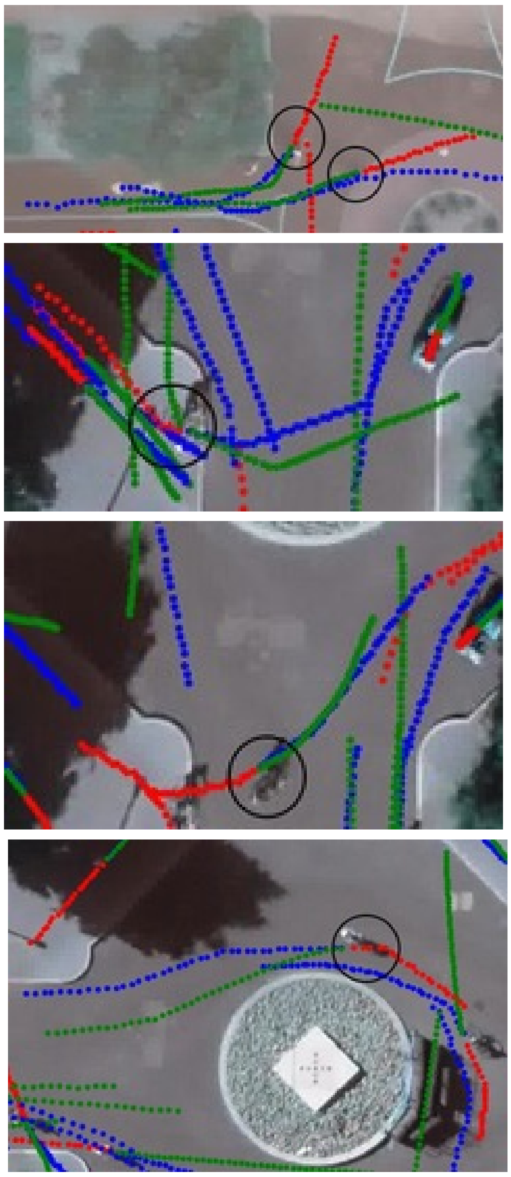
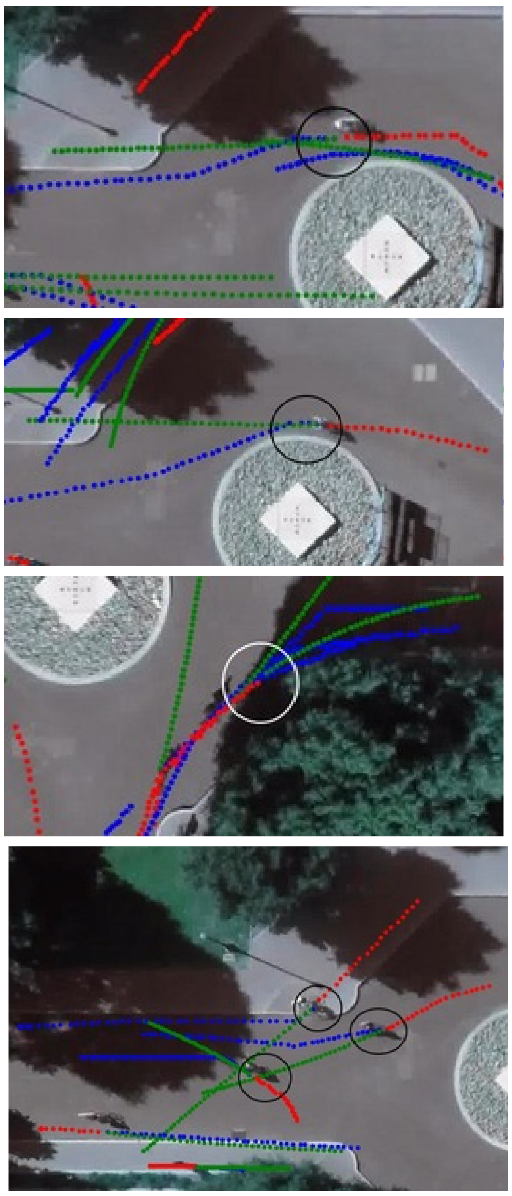

(b) RNN-ED-VSI-A

Figure 4.6: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A very well captures the scene context and interactions. The agents are predicted to move along the road direction and also without collisions. **Second row** RNN-ED-VSI-A predicts trajectory that curves along the circular junction. **Third row** RNN-ED-VSI-A is again able to capture the scene context and predicts that the agent would follow the curve along the circular junction. **Fourth row** RNN-ED-VSI-A again captures the scene context very well and predicts that the agent would move along the straight road after it has moved through the circular junction's curve.


(a) RNN-ED

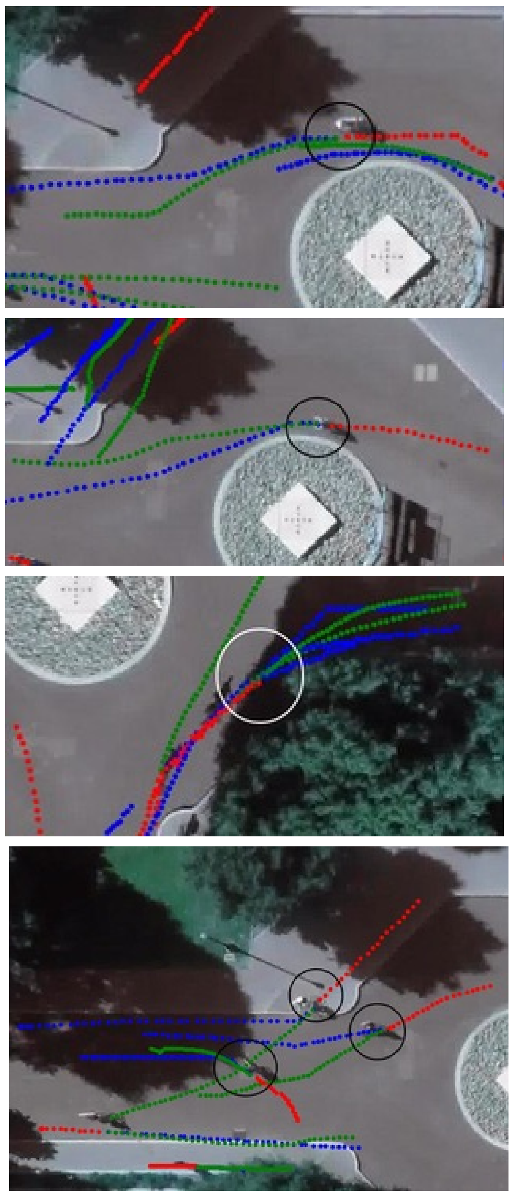
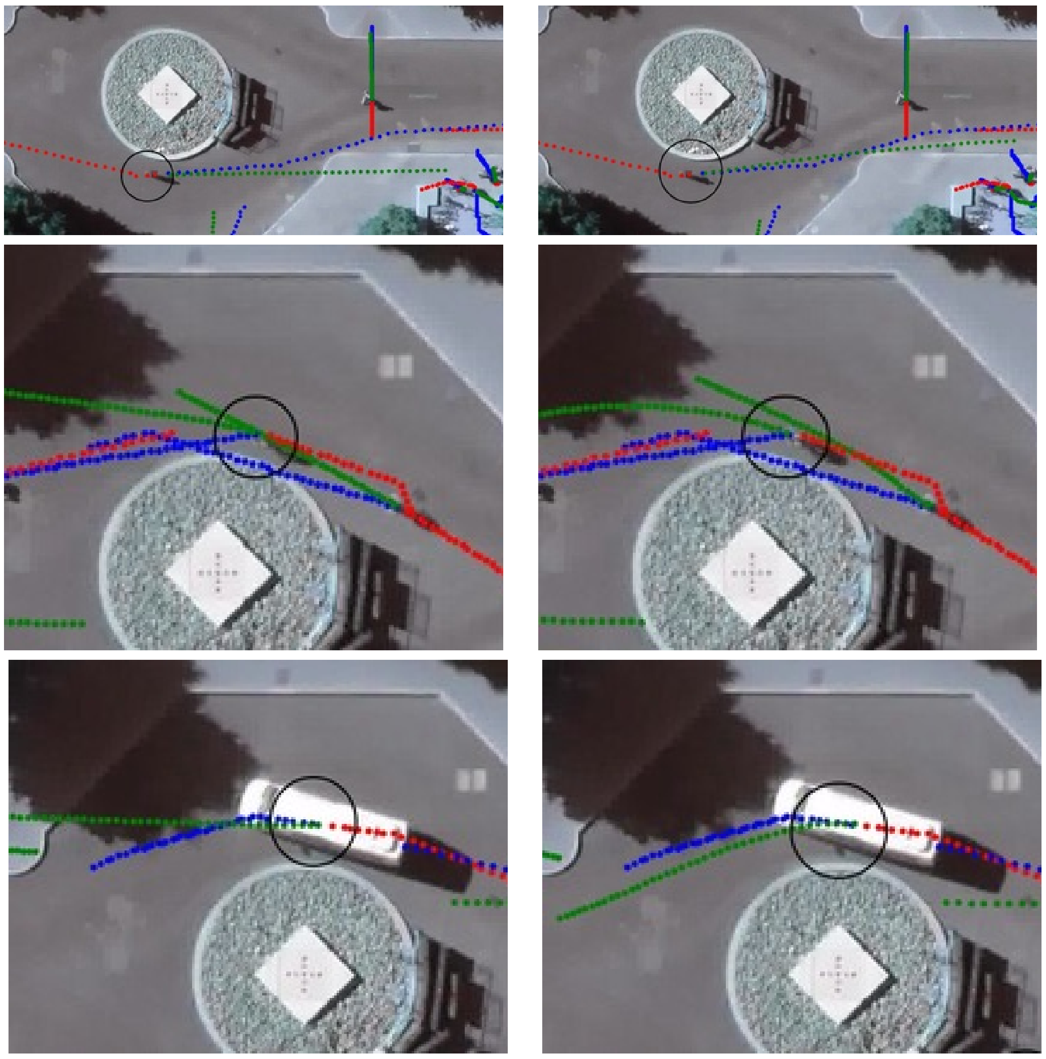
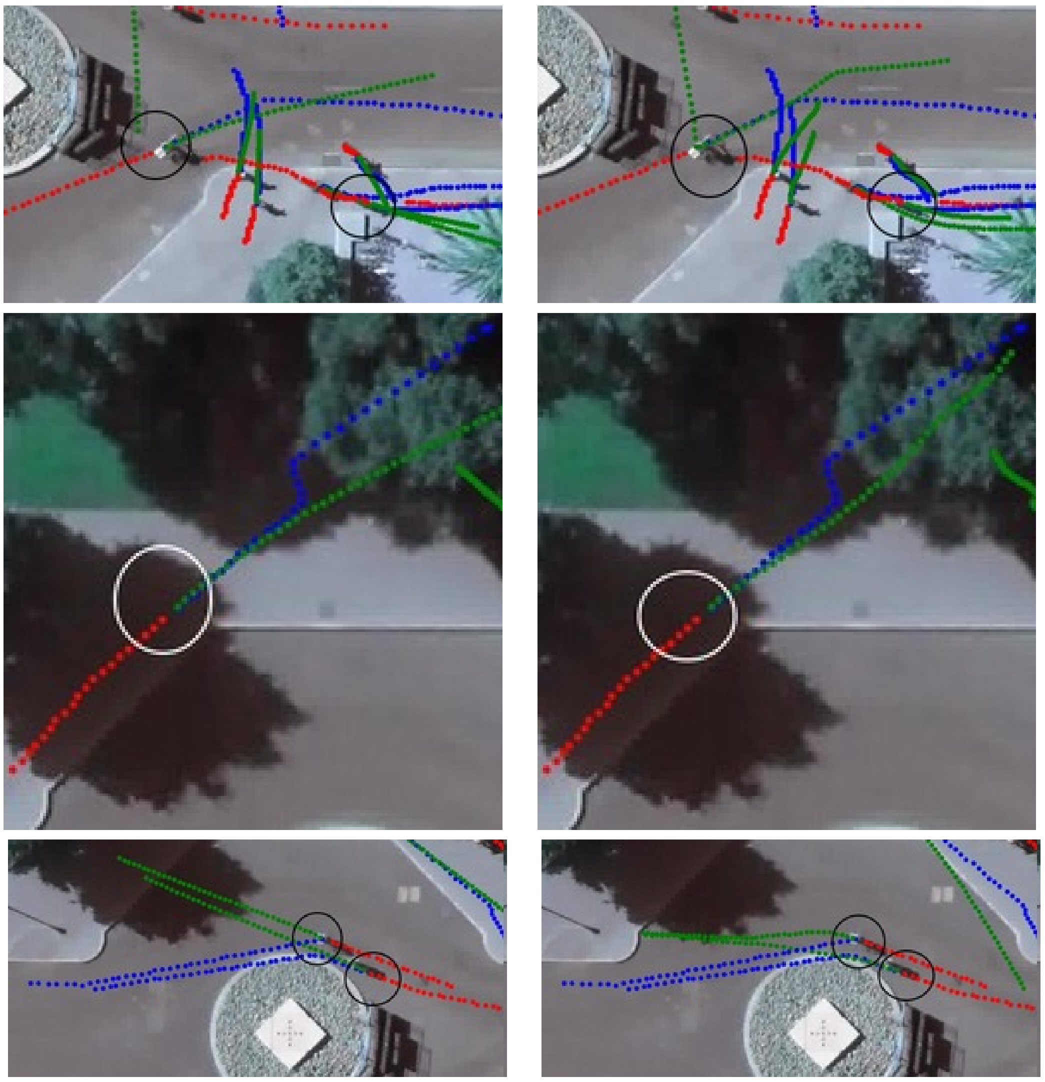

(b) RNN-ED-VSI-A

Figure 4.7: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First, Second and Third rows** RNN-ED-VSI-A very well captures the scene context. The predicted paths curve with the road. **Fourth row** RNN-ED-VSI-A very well captures the scene conext and interactions. The predicted paths are along the road directions and also such that the agents don't collide.



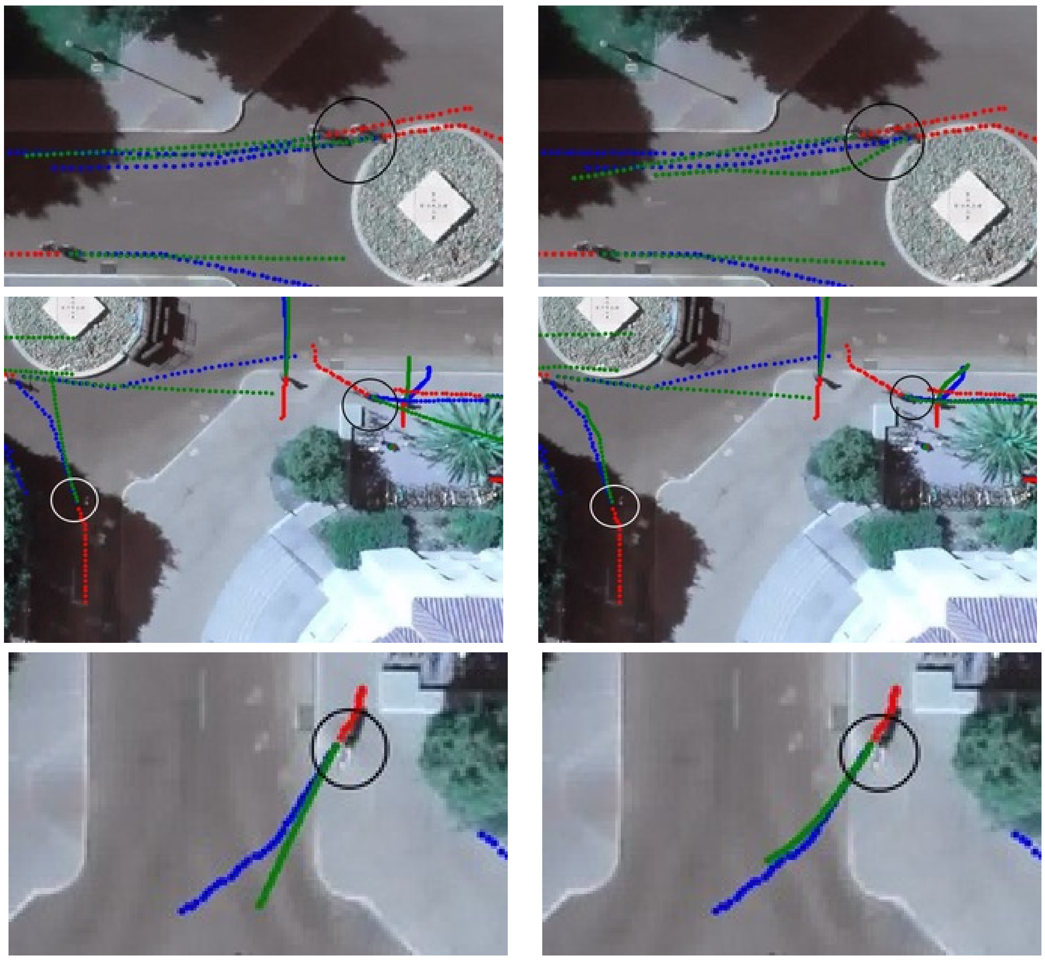
(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.8: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First, Second and Third rows** RNN-ED-VSI-A very well captures the dynamic scene context. The predicted paths curve with the road.



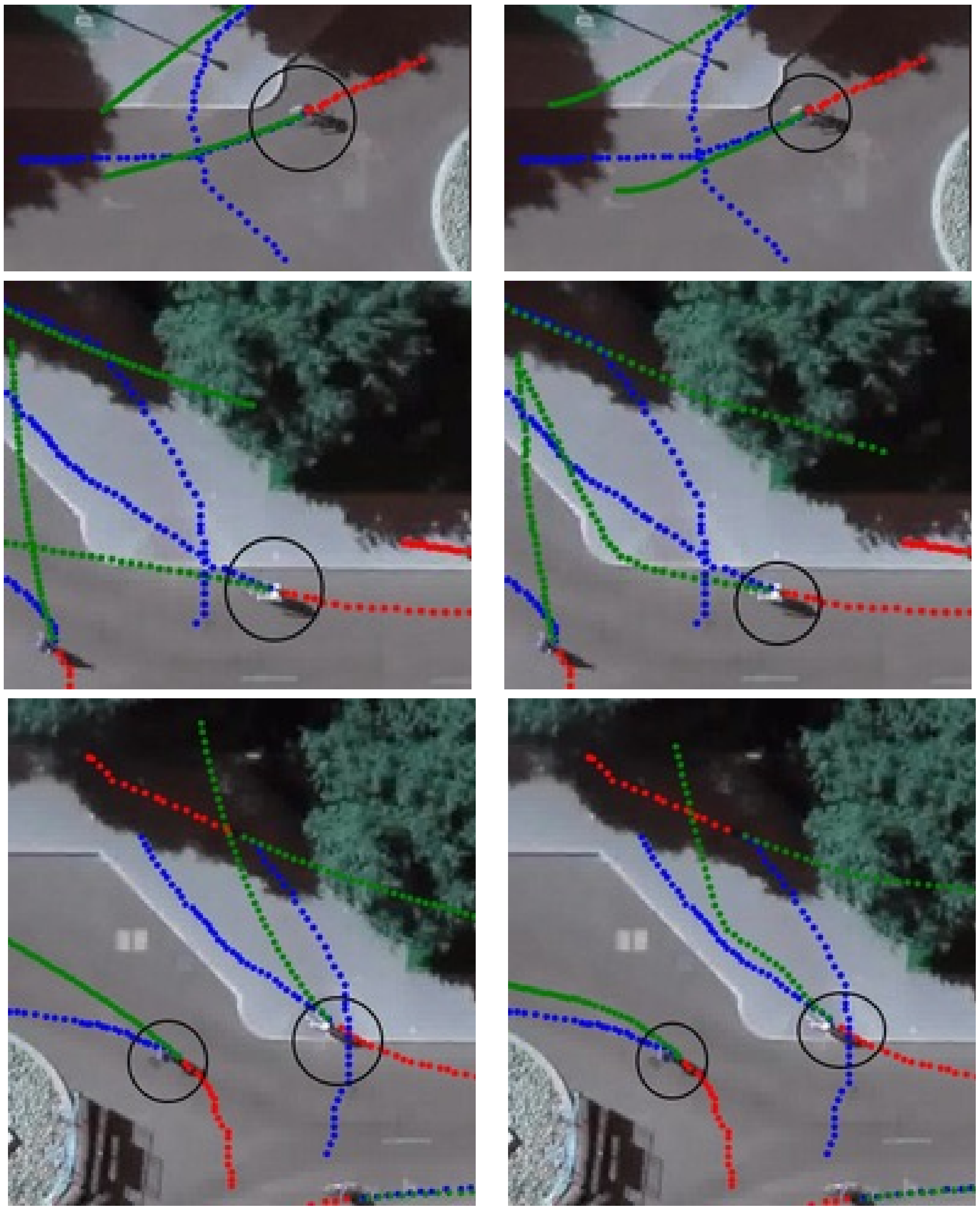
(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.9: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A very well captures the scene context. The predicted path curves along the walkable area. **Second row** RNN-ED-VSI-A is able to capture the interactions. It predicts such that the agent moves away from the neighboring agent, thus, avoiding collision. **Third row** RNN-ED-VSI-A captures the scene context and interactions. The predicted trajectory is such that the agent moves along the driveable road and also avoids collision with neighboring agents.



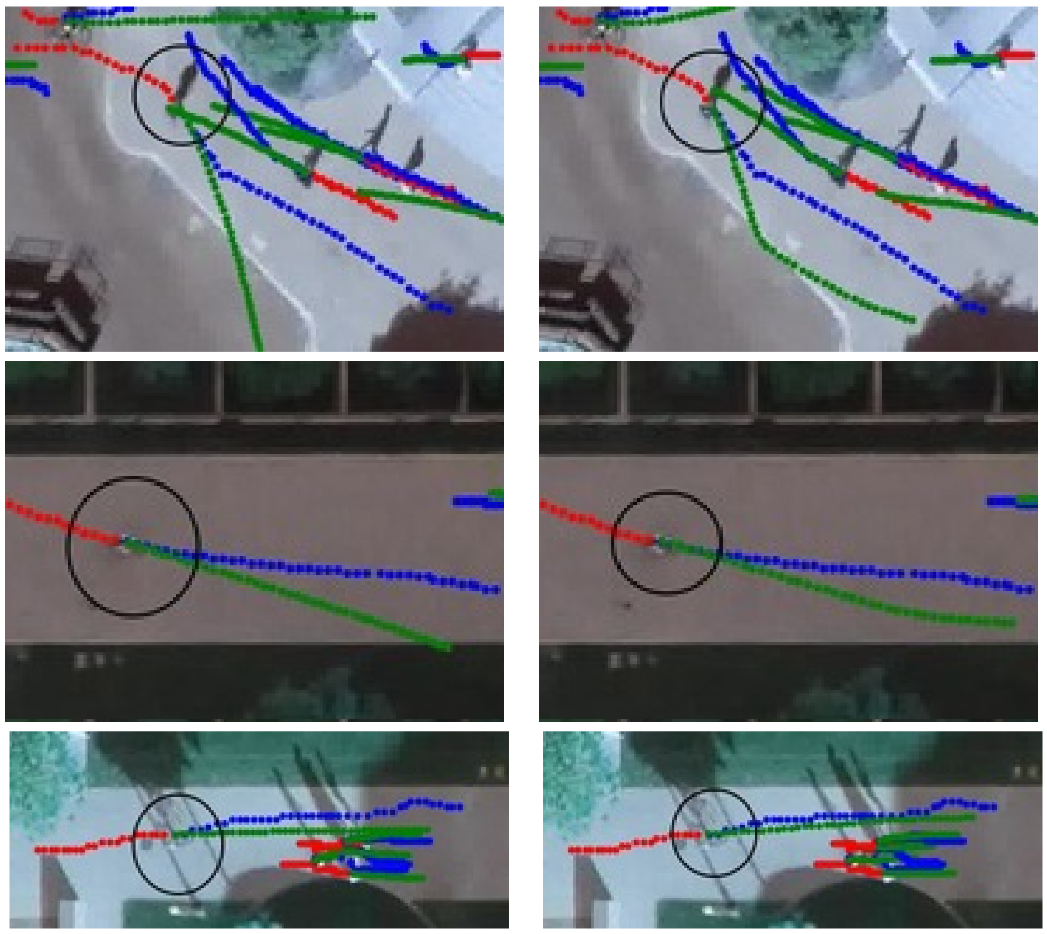
(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.10: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A very well captures the scene context and interactions. The agents are predicted to move along the driveable area avoiding collisions with each other. **Second row** RNN-ED-VSI-A is able to capture the scene context. The agent is predicted to move along the walkable area. **Third row** RNN-ED-VSI-A captures the scene context. The agent is accurately predicted to move along the curving road.



(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.11: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First row** RNN-ED-VSI-A very well captures the scene context. The agent is predicted to move along the direction of the road after exiting through the roundabout. **Second row** RNN-ED-VSI-A is able to capture the scene context. The agent is accurately predicted to take the curved road along the sidewalk. **Third row** RNN-ED-VSI-A again accurately captures the scene context. The agent is predicted to move along the sidewalk.



(a) RNN-ED (b) RNN-ED-VSI-A

Figure 4.12: Red: 2 seconds past, Blue: 4 seconds future (ground truth), Green: 4 seconds predicted future. Agent(s) under consideration represented by circle. **First and Second rows** RNN-ED-VSI-A very well captures the scene context. The agents are predicted to move along the walkable/ driveable areas. **Third row** RNN-ED-VSI-A accurately captures the interactions. All the agents are predicted to be moving in a non-colliding manner.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

With reference to experiments and results presented in Chapter 4, following conclusions are drawn:

- The model is quite accurately able to capture scene context and interactions. The agents are predicted to move along the road directions without colliding with each other.
- The model also yields highly promising performance in comparison to the most recent state-of-the-art model [38].
- Variable number of agents can be handled by our model.
- Dynamic scene is taken care of by the model.

5.2 Future Work

As future work, we would like to embed our model with generative models in order to be able to generate a family of possible future trajectories. This is necessary, since in reality we cannot be completely sure which direction an agent would move into. For example, an agent may move out into any of the exits at a roundabout. So just like humans are able to anticipate different possible trajectories with their corresponding confidence levels, the model should also be able to do the same to produce more realistic solutions.

Bibliography

[1] Cs231n convolutional neural networks for visual recognitin. `http://cs231n.github.io/neural-networks-1/`. Accessed: 2017-08-19.

[2] How does lstm cell map to layers? `https://stackoverflow.com/questions/45223467/how-does-lstm-cell-map-to-layers`. Accessed: 2017-08-19.

[3] Neuron. `http://simple.wikipedia.org/wiki/Neuron/`. Accessed: 2017-08-19.

[4] Understanding convolutional neural networks for nlp. `http://www.wildml.com/2015/11/understanding-convolutional-neural-networks-for-nlp/`. Accessed: 2017-08-19.

[5] Understanding lstm networks. `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`. Accessed: 2017-08-19.

[6] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[7] Alexandre Alahi, Kratarth Goel, Vignesh Ramanathan, Alexandre Robicquet, Li Fei-Fei, and Silvio Savarese. Social lstm: Human trajectory prediction in crowded spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[9] Lamberto Ballan, Francesco Castaldo, Alexandre Alahi, Francesco Palmieri, and Silvio Savarese. Knowledge transfer for scene-specific motion prediction. *CoRR*, abs/1603.06987, 2016.

[10] Yoshua Bengio, Ian J. Goodfellow, and Aaron Courville. Deep learning. Book in preparation for MIT Press, 2015.

[11] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-convolutional siamese networks for object tracking. *CoRR*, abs/1606.09549, 2016.

[12] Christopher M. Bishop. *Neural Networks for Pattern Recognition.* Oxford University Press, Inc., New York, NY, USA, 1995.

[13] Zhaowei Cai, Quanfu Fan, Rogério Schmidt Feris, and Nuno Vasconcelos. A unified multi-scale deep convolutional neural network for fast object detection. *CoRR*, abs/1607.07155, 2016.

[14] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L. Yuille. Semantic image segmentation with deep convolutional nets and fully connected crfs. *CoRR*, abs/1412.7062, 2014.

[15] Xiaozhi Chen, Kaustav Kundu, Ziyu Zhang, Huimin Ma, Sanja Fidler, and Raquel Urtasun. Monocular 3d object detection for autonomous driving. In *IEEE CVPR*, 2016.

[16] Xiaozhi Chen, Kaustav Kundu, Yukun Zhu, Andrew Berneshawi, Huimin Ma, Sanja Fidler, and Raquel Urtasun. 3d object proposals for accurate object class detection. In *NIPS*, 2015.

[17] Xiaozhi Chen, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. Multi-view 3d object detection network for autonomous driving. In *IEEE CVPR*, 2017.

[18] Kyunghyun Cho, Bart van Merrienboer, Çaglar Gülçehre, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *CoRR*, abs/1406.1078, 2014.

[19] Junyoung Chung, Çaglar Gülçehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014.

[20] Jifeng Dai, Kaiming He, and Jian Sun. Instance-aware semantic segmentation via multi-task network cascades. *CoRR*, abs/1512.04412, 2015.

[21] E. D. Dickmanns and B. D. Mysliwetz. Recursive 3-d road and relative ego-state recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2):199–213, Feb 1992.

[22] Ernst Dickmanns and Volker Graefe. Dynamic monocular machine vision. 1:223–240, 12 1988.

[23] A. Geiger, M. Lauer, C. Wojek, C. Stiller, and R. Urtasun. 3d traffic scene understanding from movable platforms. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(5):1012–1025, May 2014.

[24] Alex Graves, Santiago Fernández, Faustino Gomez, and Jürgen Schmidhuber. Connectionist temporal classification: Labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd International Conference on Machine Learning*, ICML '06, pages 369–376, New York, NY, USA, 2006. ACM.

[25] Alex Graves, Marcus Liwicki, Horst Bunke, Juergen Schmidhuber, and Santiago Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, editors, *Advances in Neural Information Processing Systems 20*, pages 577–584. Curran Associates, Inc., 2008.

[26] Fatma Güney and Andreas Geiger. Displets: Resolving stereo ambiguities using object knowledge. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR) 2015*, pages 4165–4175, June 2015.

[27] Zeeshan Hayder, Xuming He, and Mathieu Salzmann. Shape-aware instance segmentation. *CoRR*, abs/1612.03129, 2016.

[28] Dirk Helbing and Péter Molnár. Social force model for pedestrian dynamics. *Phys. Rev. E*, 51:4282–4286, May 1995.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, nov 1997.

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

[31] Ashesh Jain, Avi Singh, Hema Swetha Koppula, Shane Soh, and Ashutosh Saxena. Recurrent neural networks for driver activity anticipation via sensory-fusion architecture. *CoRR*, abs/1509.05016, 2015.

[32] Joel Janai, Fatma Güney, Aseem Behl, and Andreas Geiger. Computer vision for autonomous vehicles: Problems, datasets and state-of-the-art. *CoRR*, abs/1704.05519, 2017.

[33] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun. What is the best multi-stage architecture for object recognition? In *2009 IEEE 12th International Conference on Computer Vision*, pages 2146–2153, Sept 2009.

[34] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[35] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. *Activity Forecasting*, pages 201–214. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[37] N. Lee and K. M. Kitani. Predicting wide receiver trajectories in american football. In *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 1–9, March 2016.

[38] Namhoon Lee, Wongun Choi, Paul Vernaza, Christopher Bongsoo Choy, Philip H. S. Torr, and Manmohan Krishna Chandraker. DESIRE: distant future prediction in dynamic scenes with interacting agents. *CoRR*, abs/1704.04394, 2017.

[39] Philip Lenz, Andreas Geiger, and Raquel Urtasun. Followme: Efficient online min-cost flow tracking with bounded memory and computation. In *2015 IEEE International Conference on Computer Vision, ICCV 2015, Santiago, Chile, December 7-13, 2015*, pages 4364–4372, 2015.

[40] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. *CoRR*, abs/1411.4038, 2014.

[41] Nikolaus Mayer, Eddy Ilg, Philip Häusser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation. *CoRR*, abs/1512.02134, 2015.

[42] R. Mur-Artal, J. M. M. Montiel, and J. D. Tardós. Orb-slam: A versatile and accurate monocular slam system. *IEEE Transactions on Robotics*, 31(5):1147–1163, Oct 2015.

[43] Raul Mur-Artal and Juan D. Tardós. ORB-SLAM2: an open-source SLAM system for monocular, stereo and RGB-D cameras. *CoRR*, abs/1610.06475, 2016.

[44] David Pfeiffer and Uwe Franke. Towards a global optimal multi-layer stixel representation of dense 3d data. In *Proc. BMVC*, pages 51.1–51.12, 2011. http://dx.doi.org/10.5244/C.25.51.

[45] Arik Poznanski and Lior Wolf. Cnn-n-gram for handwriting word recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[46] Yao Qin, Dongjin Song, Haifeng Cheng, Wei Cheng, Guofei Jiang, and Garrison W. Cottrell. A dual-stage attention-based recurrent neural network for time series prediction. *CoRR*, abs/1704.02971, 2017.

[47] Joseph Redmon, Santosh Kumar Divvala, Ross B. Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. *CoRR*, abs/1506.02640, 2015.

[48] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. In *Proceedings of the 31th International Conference on Machine Learning, ICML 2014, Beijing, China, 21-26 June 2014*, pages 1278–1286, 2014.

[49] Alexandre Robicquet, Amir Sadeghian, Alexandre Alahi, and Silvio Savarese. *Learning Social Etiquette: Human Trajectory Understanding In Crowded Scenes*, pages 549–565. Cham, 2016.

[50] Jurgen Schmidhuber. Multi-column deep neural networks for image classification. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, CVPR '12, pages 3642–3649, Washington, DC, USA, 2012. IEEE Computer Society.

[51] Akihito Seki and Marc Pollefeys. Patch based confidence prediction for dense disparity map. In Edwin R. Hancock Richard C. Wilson and William A. P. Smith, editors, *Proceedings of the British Machine Vision Conference (BMVC)*, pages 23.1–23.13. BMVA Press, September 2016.

[52] Laura Sevilla-Lara, Deqing Sun, Varun Jampani, and Michael J. Black. Optical flow with semantic segmentation and localized layers. *CoRR*, abs/1603.03911, 2016.

[53] Ilya Sutskever. Training Recurrent Neural Networks - Ilia Sutskever - PhD thesis.

[54] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[55] Chuck Thorpe, Martial Hebert, Takeo Kanade, and Steven Shafer. Vision and navigation for the carnegie-mellon navlab. 10(3):362 – 373, May 1988.

[56] Jonas Uhrig, Marius Cordts, Uwe Franke, and Thomas Brox. Pixel-level encoding and depth layering for instance-level semantic labeling. *CoRR*, abs/1604.05096, 2016.

[57] C. Vondrick, H. Pirsiavash, and A. Torralba. Anticipating visual representations from unlabeled video. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 98–106, June 2016.

[58] Jacob Walker, Carl Doersch, Abhinav Gupta, and Martial Hebert. An uncertain future: Forecasting from static images using variational autoencoders. *CoRR*, abs/1606.07873, 2016.

[59] C. Wojek, S. Walk, S. Roth, K. Schindler, and B. Schiele. Monocular visual scene understanding: Understanding multi-object traffic scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(4):882–897, 2013.

[60] Ju Hong Yoon, Chang-Ryeol Lee, Ming-Hsuan Yang, and Kuk-Jin Yoon. Online multi-object tracking via structural constraint event aggregation. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 1392–1400, 2016.

[61] Ju Hong Yoon, Ming-Hsuan Yang, Jongwoo Lim, and Kuk-Jin Yoon. Bayesian multi-object tracking using motion context from multiple objects. In *2015 IEEE Winter Conference on Applications of Computer Vision, WACV 2015, Waikoloa, HI, USA, January 5-9, 2015*, pages 33–40, 2015.

[62] Ju Hong Yoon, Kuk-Jin Yoon, and Du Yong Kim. Multi-object tracking using hybrid observation in PHD filter. In *IEEE International Conference on Image Processing, ICIP 2013, Melbourne, Australia, September 15-18, 2013*, pages 3890–3894, 2013.

[63] Hongyi Zhang, Andreas Geiger, and Raquel Urtasun. Understanding high-level semantics by modeling traffic patterns. In *International Conference on Computer Vision*, pages 3056–3063, Sydney, Australia, December 2013.

[64] Ji Zhang and Sanjiv Singh. Visual-lidar odometry and mapping: Low-drift, robust, and fast. May 2015.

[65] Ziyu Zhang, Sanja Fidler, and Raquel Urtasun. Instance-level segmentation with deep densely connected mrfs. *CoRR*, abs/1512.06735, 2015.

[66] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network, 2016. cite arxiv:1612.01105Comment: CVPR 2017.