UNIVERSITÀ DI BOLOGNA

School of Engineering

Master Degree in Automation Engineering

Master Thesis

Trajectory Prediction for ADAS

Professor: **Luigi Di Stefano**

Student:
Muhammad Sarim Mehdi

Academic year 2019/2020

Abstract

A novel pipeline is presented for unsupervised trajectory prediction. As part of this research, numerous techniques are investigated for trajectory prediction of dynamic obstacles from an egocentric perspective (driver's perspective). The algorithm takes images from a calibrated stereo camera as input or data from a laser scanner and outputs a heat map that describes all possible future locations of that specific 3D object for the next few frames. This research has many applications, most notably for autonomous cars as it allows them to make better driving decisions if they are able to anticipate where another moving object is going to be in the future. The code is public and available here: `https://github.com/sarimmehdi/master_thesis`

Contents

Introduction **7**

1 Related Works **9**

1.1 Traditional computer vision techniques 9
1.2 Using neural networks 12
1.2.1 Recurrent Neural Nets 14
1.2.2 Variational Autoencoder 19
1.2.3 Generative Adverserial Networks 21
1.2.4 Reinforcement Learning 22
1.3 Problems with previous approaches 23

2 Methodology **25**

2.1 2D Kalman Filter 25
2.2 3D Bounding Box Propagation 28
2.3 Potential Field 33
2.4 Heatmap 39
2.5 Speed and Steering Value Recommendation System Prototype 43

3 Results **45**

Conclusions **51**

Bibliography **59**

List of Figures

1.1 Blue bounding box is the detection for the current frame, white bounding box is the predicted position of object from previous frame/s 14
1.2 GRU cell (courtesy of Wikipedia) 15
1.3 LSTM cell (courtesy of Wikipedia) 16
1.4 bi-directional RNN (Picture by Christopher Olah on his blog) 17
1.5 Global vs local attention [41] 18
1.6 VAE (Picture by Irhum Shafkat) 19
1.7 GAN (Picture by Emiliano De Cristofaro) 21
1.8 RL Agent plays Atari games (Picture by Ahmet Salim Bilgin) 22

2.1 KITTI with Kalman Filter Example 1 (blue is the current ground truth bounding box and green is the Kalman filter prediction) 26
2.2 KITTI with Kalman Filter Example 2 (blue is the current ground truth bounding box and green is the Kalman filter prediction) 27
2.3 3D Bounding Box Propagation 28
2.4 KITTI with 3D bounding box propagation upto a single level of recursion (pink is the current bounding box and yellow is the prediction) 32
2.5 top most picture shows the 3D bounding box in the original image, middle picture shows the bounding box in the image plane with only the semantically labeled pixels shown (for road and pavement), and the bottom picture shows the same situation from a slightly elevated perspective with only point cloud present 33
2.6 Now the bounding box is pushed forward in space upto a maximum limit and all valid positions are selected based on the IoU of the base of the projected 3D bounding box (the one being pushed forward from its original position) with the road pixels (which were semantically labeled with the help of a neural net). We can see that the proposed bounding box has a very low IoU because there are other cars parked in front 34

2.7 In order to look for better proposals, the bounding box is rotated in the anticlockwise direction by 10 degrees (it is rotated upto a prescribed maximum angle in the anticlockwise direction from its current heading angle and then rotated upto the same prescribed maximum angle in the clockwise direction) . 35
2.8 Now, better proposals are obtained since the new forward-projected bounding box has a really high IoU with the road . 36
2.9 A radial potential field growing outward (for each top-bottom pair, the bottom picture represents the potential field in the point cloud domain and the top picture represents the same in the image plane) 38
2.10 A heatmap that grows outwards in the form of a 60 degree cone, it is hot close to the object but gets colder as you move far away. Clearly, this is visually appealing and more informative than all the previously mentioned approaches 40
2.11 Probability that an object might turn left, turn right, or go straight (calculated using the heatmap) 42
2.12 Speed and steering recommendation based on 3D bounding box and its heatmap 44

3.1 Heatmap of inaccurate 3D bounding box is guided by the panoptic segmentation 46
3.2 Heatmap of inaccurate 3D bounding box is in the wrong lane 47
3.3 Inaccurate panoptic segmentation, at night-time, has an effect on the heatmap generation 48
3.4 Block diagram representing the entire process 49

Introduction

Self-driving cars are getting better every year. Full autonomous driving capability (Level 5, meaning no human driver needs to be present at all) is still far off but the progress we have been witnessing is very encouraging. As such, many self-driving cars make use of reinforcement learning algorithms that take their environment as input using a diverse sensor suite. The output is usually a steering angle and speed value. A big problem is avoiding obstacles and maneuvering around them. This gets even more challenging in densely packed urban areas. The problem is that even if the self-driving car alone makes all the right decisions and abides by all traffic rules, everyone around it cannot be expected to do the same. You can have pedestrians trying to cross the road when they shouldn't, other vehicles taking unexpected turns or behaving in an unusual way (braking all of a sudden etc.). One must remember Isaac Asimov's First Law of Robotics when deploying a self-driving car in such a scenario: "A robot may not injure a human being or, through inaction, allow a human being to come to harm."

Most reinforcement learning agents only consider the current position of all dynamic obstacles when making driving decisions. This is not appropriate and, in fact, we as humans do our best to anticipate where others will be in the future when we are driving. It only makes sense for a self-driving car to do the same. As such, making accurate predictions about other moving objects is a very hard problem to solve in practice. Attempts to model the motion of moving objects in a social context go as far back as 1998 [25]. Here, a social force model is presented which can be extended to other areas like opinion formation and group dynamics. The force is simply the sum of all the attractive (friends, street-exhibitions etc.) and repulsive (strangers, obstacles etc.) forces with an extra term to take into account the fluctuations (to model uncertain behavior like which path to follow when navigating around an obstacle and so on). A more discretized approach was adopted in [5] where each pedestrian has a 170 degree cone in front of them and the cone is divided into three regions: Deceleration, Constant Speed and Acceleration. The behavior of each pedestrian is influenced by other pedestrians. However, this time, many parameters were introduced by the authors and these had to be estimated to observe different kinds of behaviors.

Motivations

The purpose of this research is to find an unsupervised approach towards prediction that can, later on, be used in conjunction with a self-driving car to help it make better decisions. Any given scenario can have multiple different futures. To understand why this is the case, take the example of a situation in which a person is standing on one side of the road and facing it. An observer might say that there is a 50-50 chance of the person crossing and not crossing the road. Another observer might say there is a 60-40 chance of that happening.

The scope of this research is to provide a prediction 'heatmap' using a robust 3D object detection and prediction pipeline that can work with images from a stereo camera or data from a laser. When working with stereo images, a depth map is generated and this is converted to a Point Cloud using the Pseudo LIDAR [68] approach.

Organization

The structure of the Master thesis is as follows. In the next section, relevant research literature is presented and there is a discussion about how it tried to tackle the problem of prediction in many different contexts. After that, our approach is presented and its qualitative results discussed. Finally, the research concludes with closing remarks and possible future research questions that can be discussed as a consequence of this Master thesis.

Chapter 1

Related Works

Literature that deals with the problem of trajectory prediction can be divided into two portions: Using traditional image processing techniques and using neural networks. Both approaches are unique and deserve to be discussed in full to understand how the problem of prediction (and not just trajectory prediction) has changed over the years.

1.1 Traditional computer vision techniques

Computer vision approaches towards predicting abnormal behavior were used in [43]. Here, the Social Force model is employed by overlaying a grid of particles on a video. The interaction forces between the particles for every frame are used to model the crowd and this is modeled as a force field. This is combined with LDA (Latent Dirichlet Allocation) to distinguish between normal and abnormal video frames. [67] looks at video surveillance footage for intersections and roads and makes a prediction about abnormal activities, single-agent activity prediction (whether a car would make a U-turn for example), and discover interactions which might take place in the future between traffic participants. The authors used hierarchical Bayesian models to connect low level image features (pixel-level), atomic activities (car driving etc.) and interactions between different video sequences (since activities taking place in different video clips have underlying distribution which can be linked together). The models are learned in an unsupervised manner and can describe long video sequences. [47] uses an energy function to predict future velocities of pedestrians using images obtained from a bird's eye view. The model takes into account static as well as dynamic obstacles (other pedestrians) and velocity prediction is done by minimizing the energy function using a gradient descent approach with line search.

[79] does event prediction as well as telling us how unusual the predicted event is. They use brief video clips and model each using trajectories of key-points over time. 2000 salient points are obtained for the first video frame

and their trajectories computed using the KLT (Kanade-Lucas-Tomasi) algorithm. The tracker seeks the distance that gives the minimum dissimilarity for a given window around a selected point between two consecutive frames. A lost point (due to occlusion or severe luminosity changes) is replaced by a new one. Each point is tracked and clustered with others using a distance metric chosen by the authors and a 2000 by 2000 affinity matrix. The authors used a fixed number of clusters (10). For each cluster, a velocity histogram is computed (a regular grid with a cell spacing of 10 pixels is put on top of the image frame to create a spatial histogram containing 8 subbins at each cell in the grid) and the similarity between two clusters is given by the intersection of their velocity histogram. This new metric is supposed to represent spatial coherence and, therefore, videos without any unusual event would give good values with respect to the ground truth video. The authors also try to do event and anomaly prediction using a single image by applying the above algorithm on the features obtained via SIFT and GIST.

[36] treats the problem of anomaly detection as a statistical model where an unusual event would have a very low likelihood of occurring based on some distribution under *normal* circumstances. An event is described temporally and spatially. So, a normal event would happen continuously and happen at the same location whereas it would be the opposite for unusual events. A generative model for the MDT is defined and the parameters are estimated using maximum likelihood from a collection of video patches. The method looks for temporal and spatial abnormalities.

In [63] the authors model the spatial scene using a Gaussian Mixture Model (GMM) in the training phase. The GMMs are fit to the training data using the Expectation Maximization (EM) algorithm. Learning of the scene structure is performed in an offline training phase and is not updated online during system operation. To encode information about travel patterns between entry, turning, and exit regions, a transition model is learned in terms of region-to-region segments. This is used to predict the next path segment the agent will choose upon arriving at its destination. The prediction relies on the Markov assumption: The agent's current state depends on a finite number of previous states. The authors decide to only go as far back as the immediate predecessor of the agent's current position. Afterwards, a particle filtering algorithm is used to classify trajectories in two stages. In the prediction step, the particles are propagated forwards according to the state transition model $p(x_t|x_{t-1})$. In addition, a weight or importance factor is computed for each particle. The weight of a particle is given by its observation likelihood $p(z_t|xt)$. Next, in the correction step, the observation is incorporated by resampling the particles based on the computed weights. This learned transition model is used to predict whether a particle trajectory is unusual or not during live-execution.

[46] defines atomic actions and predicts the intent of agents using a top down tree parsing method. The algorithm takes as input several parse

graphs and event parsing is done by defining atomic actions at each frame of the video. The parse graph that best explains the events happening up to a given frame of the video is chosen and the future event is then predicted using that graph.

[74] uses an energy function to model the speed, direction, collision probability, grouping (pedestrians belong to a group based on how close they are to each other and which direction they move in throughout the video) and attraction. The overall energy function is a weighted sum of energies for each of these properties and this is minimized to find the desired future velocity of each pedestrian which is used to predict future positions of each person several frames into the future. Parameters for the energy function are estimated by finding the local minima of a complicated convex problem. The authors also manually annotated the data for it to be used according to their method (defining desired velocity for each pedestrian, its final destination, and social group based on a histogram of positions and velocities).

[31] predicts multiple trajectories for any given agent and assigns a probability to each trajectory. This is achieved using Inverse Optimal Control where a reward function is learnt using demonstrations and that is used to determine the optimal policy which describes the most likely path an agent would take if it starts at a certain location. Their algorithm also takes into account error due to a noisy tracker. At train time, a grid is overlayed on the video sequence and IOC is applied and the reward function (which is a weighted sum of features, where a feature describes the ease of travel) determined. This is used at test time to get multiple trajectories.

[83] uses a new MDA (Mixture Model of Dynamic Pedestrian Agents) to learn the collective behavior of a crowd using surveillance footage. Each agent has its own dynamics and beliefs. Dynamics are modeled as a set of discrete system equations with a state transition matrix, observation matrix, system noise and observation noise. Beliefs are modeled using Gaussian Distributions with certain means and covariances. The parameters of dynamics and beliefs are learnt using the EM algorithm (with training video footage) and then the system is used, at test time, to predict the past and future trajectories of pedestrians in new video sequences.

[32] uses a DBN (Dynamic Bayesian Network) for pedestrian path prediction. The pedestrian's decision to stop or continue crossing the road depends on the approaching vehicle, their own awareness and their position from the curb. These three factors are captured as latent states by a DBN on top of an SLDS (Switching Linear Dynamical System) which outputs the probability related to one of three actions: SV (saw vehicle), SC (situation control) and HSV (has seen vehicle). [66] uses an unsupervised approach to generate a prediction heat map. Mid-level features from a static image are extracted using a sliding window approach. A detected feature has a transition probability for every new state (a car turning left, right etc.) and this

probability is also influenced by other moving agents in the scene as well as the semantics of the scene (whether a car is likely to move on the road or get on the sidewalk). The interaction between different agents depends on a reward function which is learned in an unsupervised manner. During the training phase, the mid-level representation and transition probabilities are learned using large quantities of spatio-temporal data.

[3] uses SAM (social affinity maps) to make predictions about crowd movements. The idea is to represent subconscious social interaction between people (the leader-follower phenomenon is an example of this, where people tend to form lanes in densely packed areas as the fast pedestrians start passing the slower ones) using discrete maps which are represented as radial histograms using Gaussian Mixture Models. Crowd forecasting is done by creating a Markov chain where you have probabilities defining the likelihood of transitioning from one intermediate tracklet to the next one. Then, an integer linear programming problem is solved to obtain a solution which identifies the trajectory to which a certain intermediate tracklet belongs to (and this can be followed all the way from the origin tracklet to the destination tracklet). [10] assumes that every person will try to follow the path with the least amount of resistance in order to get to their destination. They define a time surface (every node on this has a value that tells how long it takes a person to reach it) and a minimal action surface (defined by the number of people frequently crossing it within a given time frame). The nodes in both surfaces are updated using a heuristic and then a final path is computed for each agent with a probability associated to each path.

[76] uses energy maps to model pedestrian behavior and predict their trajectory based on that. A particular region is a combination of higher energy (easy for people to walk through it) or lower energy (the exact opposite of higher energy). Lower energy areas are usually located close to other pedestrians and stationary obstacles. While generating a path, repulsive forces due to stationary obstacles and other groups of moving people are modeled and taken into account.

1.2 Using neural networks

So far, the approaches we have discussed make use of loosely defined heuristics and then parameters in those are either learned through training examples, in real-time, or in an unsupervised manner. There are obvious issues with this approach. One being that they are only applicable to that particular version of the problem. Furthermore, many of the techniques discussed so far use surveillance footage with a static background. Real-life applications of prediction almost always use non-static background (trying to predict motion of objects while the ego-object, from whose perspective you are making the prediction, is itself in motion).

Neural networks have proved to be much better at capturing the non-linear nature of problems like these. An example is the SORT (Simple Online and Real-Time Tracking) algorithm [7] which tries to track the same object by predicting its future location based on a Kalman Filter. The tracklets are obtained using an object detection algorithm like YOLO (You Only Look Once) [49] and a constant velocity Kalman filter is used to predict their future location. The intersection over union between the predicted bounding box and the detected bounding box (obtained from the YOLO neural net every frame) is used as an association metric followed by the application of the Hungarian algorithm to decide which predicted bounding box represents which current detected object based on the IoU. This method was still not as robust because an object that disappeared behind another object and reappeared after a few seconds would be assigned a new ID even though it was the same object. To cope with occlusions, Deep SORT [71] was introduced. The idea was to now use the cosine similarity between extracted feature vectors from the detected and predicted bounding box as an association metric as well as the IoU between predicted and detected bounding boxes. This made tracking more robust to brief occlusions. However, the previous two popular tracking algorithms cannot be used to predict the position of detected objects too far into the future. This is because, as time goes on the shape of the object and its texture can change quite a bit (due to change in light intensity etc.). This is why, in these algorithms, the track IDs are checked every frame instead of waiting for a few frames before checking to see if the same object arrived at a given location in the image plane.

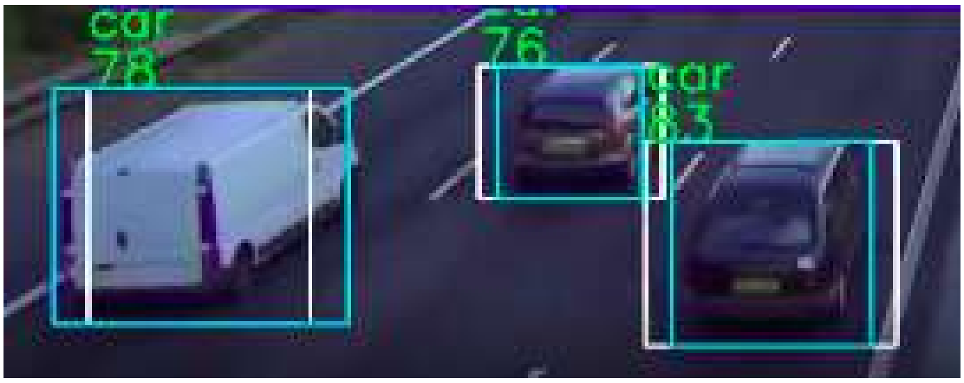

Figure 1.1: Blue bounding box is the detection for the current frame, white bounding box is the predicted position of object from previous frame/s

1.2.1 Recurrent Neural Nets

Many have also tried to solve the prediction problem using RNNs (Recurrent Neural Nets) [22]. RNNs have gained popularity in recent years because of their usefulness in time-series prediction. An RNN consists of numerous cells that take input at each step. Each cell has a hidden state which is usually kept zero for the first cell and then updated in a non-linear fashion using input at the current step and the hidden state from the previous cell. The idea is to encode a sequence and then decode it to predict future values. The encoder consists of numerous cells that take the input and update their hidden state all the way till the last cell. The last cell then provides its hidden state as input to the decoder which repeats the same process as before. Only this time, the input to each cell after the first one is the output of the previous cell. RNNs can be of four types: Many-to-Many, One-to-One, One-to-Many, Many-to-One. Usage of each type depends on the task. Many-to-Many can be used in speech recognition where the input is an audio signal and an output is a sequence of words which the RNN believes the person said [14]. One-to-Many can be used in image captioning where the input is a picture and the output is a sentence describing what is happening in it [64]. Many-to-One can be used for activity recognition where the input is a video and the output is one of many possible actions [58]. On the other hand, One-to-One is simply a generic neural network without the hidden layer.

RNNs have found success in text classification [39], text translation [72], text completion [20] and even text generation [23]. They have also been used for image captioning [28]. A typical RNN cell is either a GRU (Gated Recurrent Unit) [15] or LSTM (Long Short Term Memory) [61]. The two have different architectures and mainly deal with the vanishing and exploding gradient problem for RNNs. The reason is that due to numerous multiplication operations, the gradient value might either get really big or small by the time it backpropagates all the way till the first RNN cell. There are also versions of the above two that do convolution instead of simple matrix dot product. These are useful for encoding temporal features for a video sequence [57]. The main difference between LSTM and GRU is in the number of gates. GRUs are less computationally expensive than LSTMs due to having fewer gates. However, there is very little difference in performance between both.

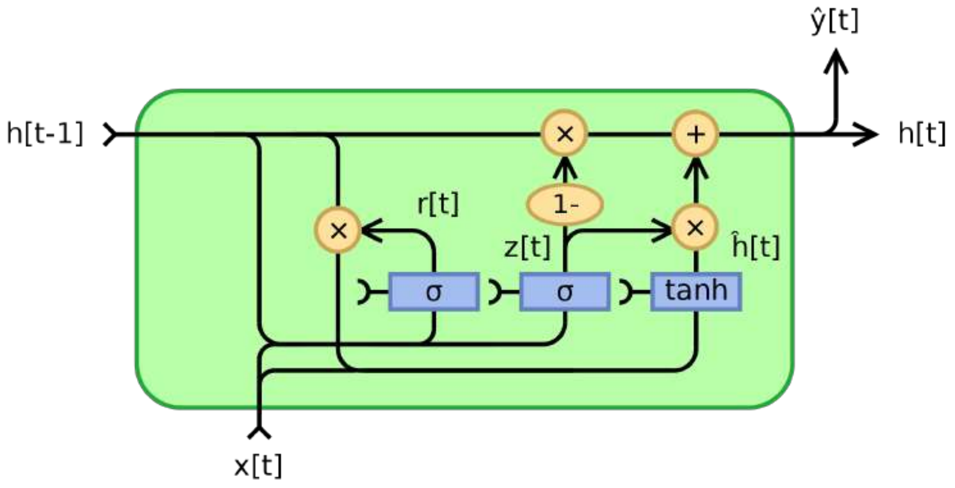

Figure 1.2: GRU cell (courtesy of Wikipedia)

$$
\begin{aligned}
z_t &= \sigma_g(W_z x_t + U_z h_{t-1} + b_z) \\
r_t &= \sigma_g(W_r x_t + U_r h_{t-1} + b_r) \\
h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \phi_h(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)
\end{aligned}
$$

x_t: input vector
h_t: output vector
z_t: update gate vector
r_t: reset gate vector
W, U and b: parameter matrices and vector
Activation function/s:
σ_g: The original is a sigmoid function.
ϕ_h: The original is a hyperbolic tangent.
$\odot$ denotes the Hadamard product

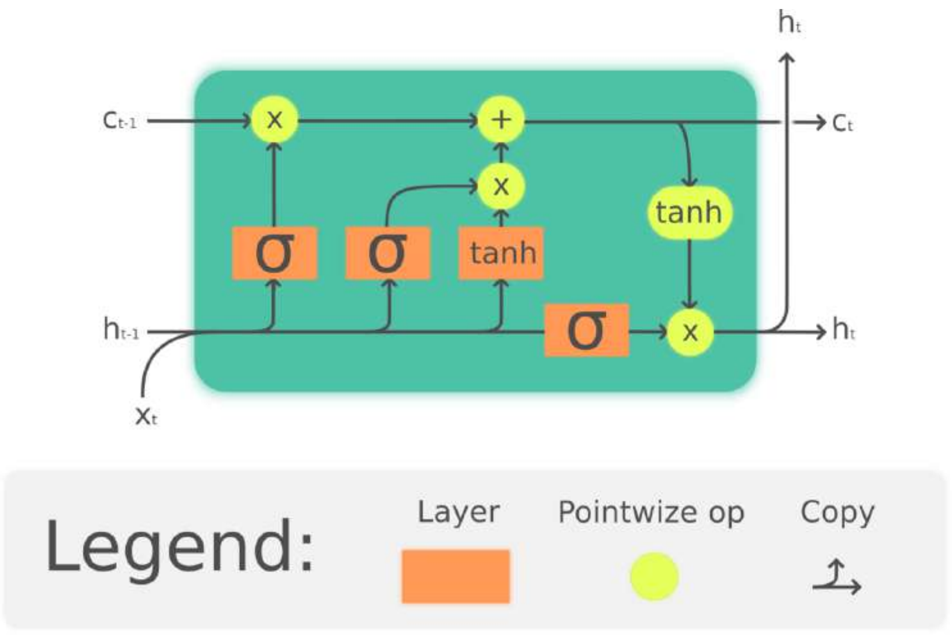

Figure 1.3: LSTM cell (courtesy of Wikipedia)

Where the initial values are $c_0 = 0$ and $h_0 = 0$ and the operator $\circ$ denotes the Hadamard product (element-wise product)

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

$x_t \in \mathbb{R}^d$: input vector to the LSTM unit
$f_t \in \mathbb{R}^h$: forget gate's activation vector
$i_t \in \mathbb{R}^h$: input/update gate's activation vector
$o_t \in \mathbb{R}^h$: output gate's activation vector
$h_t \in \mathbb{R}^h$: hidden state vector
$\tilde{c}_t \in \mathbb{R}^h$: cell input activation vector
$c_t \in \mathbb{R}^h$: cell state vector
$W \in \mathbb{R}^{h\times d}, U \in \mathbb{R}^{h\times h}$ and $b \in \mathbb{R}^h$: weight matrices and bias vector parameters which need to be learned during training

There are different types of RNN structures as well. An example is a bi-directional RNN [52] where the encoder consists of a forward and backward recurrent layer. Each recurrent layer has the same number of GRU cells and each cell at a specific step has the same input but the final output at that step is formed by concatenating the output of both cells. In the forward layer, the hidden state travels forward from the first step till the last one while in the backward layer, the hidden state travels in the opposite direction. The reason for doing this is to take advantage of the past and future when encoding an input. For example, if you encode an image sequence in one direction only then the hidden state of each cell only encodes information from the past. However, using a bi-directional RNN, allows us to combine information from the future as well. As a result of this, each step now contains information from both the past and future which captures the context of the whole video in a more temporally-appropriate manner. This makes sense because a simple image sequence of a car driving on the highway could have multiple possibilities (car might stop, it might take a turn etc.) and encoding this information in a uni-directional manner doesn't fully capture the context of this image sequence as opposed to using a bi-directional RNN.

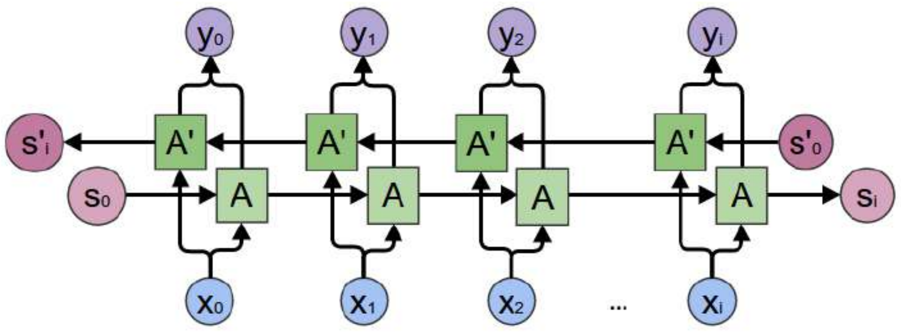

Figure 1.4: bi-directional RNN (Picture by Christopher Olah on his blog)

Another type of RNN makes use of the attention mechanism [6] when decoding to give the final output. At every step of decoding, a context vector is used to compute the hidden layer at that step as well. This context vector is computed as a weighted sum of all the encoder hidden states, where each weight is computed using a non-linear function followed by softmax. An example of combining bi-direction RNN with ConvGRUs and the attention mechanism can be found here [24]. Here the authors predict feature maps instead of the action itself. As a result, the encoder is trained to extract a feature map from a sequence of images which is used as input to the decoder. The encoder is trained to provide a feature map similar to the one obtained by applying a temporal convolution to the input at the decoder step. Backpropagation takes place without using the actual output (the action label which the authors are trying to predict). Hence, this is an example of self-supervised learning. After training the encoder and decoder like this, a fully connected layer is put on top of the decoder and this is trained in a supervised fashion to predict the right action label for the video sequence.

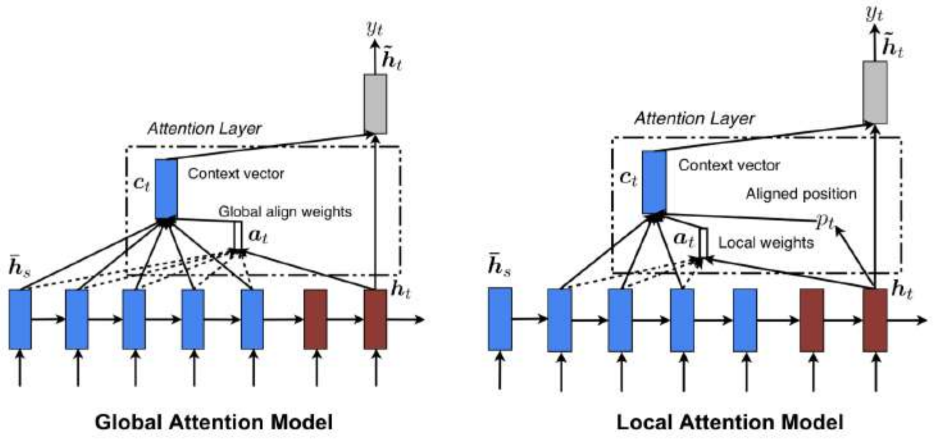

Figure 1.5: Global vs local attention [41]

[27] uses a sequence of driver facial expressions and outside information (street maps and GPS information) to anticipate one of many possible discrete driving behaviors (turn left, turn right etc.) Features from inside the car (facial features and their trajectory obtained using KLT) and outside the car are extracted and used as input to an LSTM-RNN at each step. The goal is to predict the driving actions with as little temporal context as possible.

[82] used a birdseye perspective to predict the trajectory of multiple vehicles driving on a road or highway. An LSTM encoder is used to extract a feature tensor from each vehicle using position values. This is then indexed into a higher dimensional space based on the most recent position of each vehicle. This higher dimensional space is passed through a U-Net to encode spatial relationships between the different moving agents. The decoder is then applied to this higher dimensional space and future position values for each vehicle are predicted.

[4] applies LSTM on each mobile agent followed by social pooling of the hidden layers of every agent at each time step. At each time step, a hidden state tensor is defined with dimensions N_oxN_oxD. Where D is the depth of the tensor (equal to the length of the hidden state vector) and the length and width of the tensor correspond to the size of the nearby neighborhood. The hidden state vectors are sliced into this bigger tensor and then pooled before being passed to the next LSTM unit in the RNN.

[42] try to solve the problem of prediction from motion capture data. They use highway connections to directly link the input of a decoder GRU to the output before giving the final prediction. According to the authors, doing this allows their predictions to be more fluid, meaning that there is no

discontinuity between consecutive predictions. They compare their results using previous state of the art work that only used an LSTM RNN with no highway connections and, thus, suffered from breaks and jumps between consecutive predictions.

1.2.2 Variational Autoencoder

Another popular architecture for trajectory prediction using deep learning is the variational auto encoder. A simple autoencoder [30] is a self-supervised neural net where the input and output have the same dimensions. The encoder squeezes the input to a lower dimensional latent representation using multiple convolutions which is then expanded using a decoder (deconvolution where empty entries are treated as zero when computing values that will be padded to the original feature map). The output of the decoder is compared to the original input using a loss function and backpropagation takes place to update the parameters in such a way so that the decoder learns to generate the input as accurately as possible. One application of this is in denoising images because since the auto-encoder already knows what the data is supposed to look like, it can take corrupted data as input (noisy images are fed but the loss function is computed with respect to the clean image) and output a cleaner version of it with almost all the noise removed. The variational autoencoder not only does dimensionality reduction but can actually recreate different versions of the input data. In a normal autoencoder, the latent space is directly connected to the decoder and hence only one kind of output is possible. However, in VAEs, the latent space is sampled. Sampling is done by randomly choosing a value from a Gaussian distribution with a mean and variance defined by the latent representation vectors. Running backpropagation through sampled neurons is achieved using the reparamatrization trick.

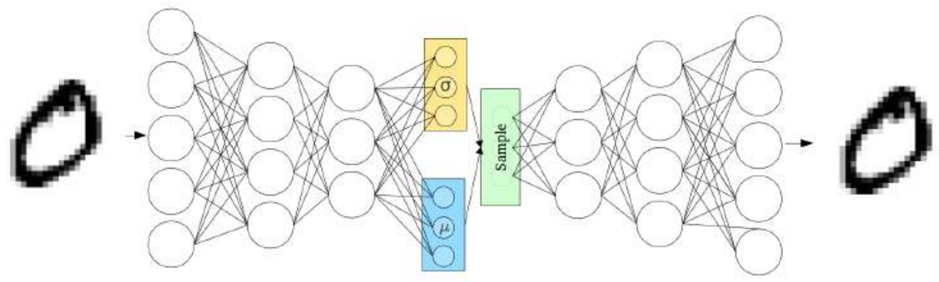

Figure 1.6: VAE (Picture by Irhum Shafkat)

[65] uses a VAE to jointly encode an image with the trajectory of each pixel in it. During training time, the trajectory of each pixel is obtained by tracking each pixel's location, from its original location in the first frame,

along the entire length of the ground truth video. To train the VAE, the image and trajectories both are encoded together. [84] experiment with VAEs and LSTM recurrent nets to generate a picture of what the object would look like in the future based on the first few frames. First they experiment with a VAE where they encode and decode a sequence of images but, alongwith the latent vector, there is also a one-hot vector which decides how far into the future should the new representation be. Then they try the same VAE architecture as before, only this time there are two separate encoder stages. One gets an image at time step t and the other gets an image at time step t+m and the latent representation of both encoders is used for the decoder stage. Finally, they check the performance of a simple LSTM RNN encoder combined with a decoder. [9] uses three different kinds of autoencoders to predict human motion given mocap data. First they use a simple VAE on the entire sequence of mocap data. Then they encode the input data using temporal convolution (convolution in time, meaning previous k data points are convolved with the same filter and the k points before those are convolved with a separate filter and so on). And finally, they try hierarchical convolution where each body part is treated as a part of a larger group and temporal convolution on that part is done separately.

[59] uses semantic and instance segmentation combined with depth maps and LIDAR point clouds to predict future trajectories. A convolutional encoder-decoder architecture with bridge connections between similar scales on both ends is used to make the final prediction. Their method transfers to other datasets quite well, meaning that they could train on the KITTI [19] dataset and then use the pretrained network on Cityscapes [16]. [38] utilizes semantic segmentation to not only predict future trajectories of all participating agents but also to predict future activities. A person's visual and pose features are extracted and used to model any changes in their behavior. A person interaction module is used to encode the interaction a person has with their surroundings which are semantically labeled using a pre-trained neural net. A Person-Object and Person-Scene Encoder provide a feature tensor which is combined with the previous encoded features to generate a trajectory and predict future activity as well. [81] uses LSTM with state refinement to encode and decode multi-object trajectories for prediction purposes. The state refinement module is trained to pay more attention to the LSTM feature vector of some pedestrians as opposed to others. Hence, this is an example of a macro attention mechanism only this time the final context vector is generated by looking at the final cell state of each LSTM encoder. [73] breaks down the problem of crowd motion prediction into three separate modules. The location encoder module extracts feature vectors from the most recent pedestrian positions. A motion encoder module extracts feature vectors from an entire sequence of positions leading up to the most recent one. The feature vectors from both modules are combined in a separate crowd interaction module before being used to generate

a displacement vector that is simply added to the most recent position of each pedestrian. [8] used two separate RNN Encoder Decoder architectures to predict future car steering and speed values and pedestrian bounding boxes. The decoding process is aided with a separate CNN that extracts a feature vector from the entire image sequence for this exact purpose.

1.2.3 Generative Adverserial Networks

Generative Adverserial Networks (GANs) [21] consist of two competing neural nets: Generator and Discriminator. The Generator takes a randomly sampled noise vector as input to generate a fake sample which is then compared with the actual ground truth to check if both are the same or not. The latter task is accomplished by the Discriminator and the resulting backpropagation is carried out all the way to the Generator. So, in essence, both the Generator and Discriminator compete with each other. The Generator is trying to fool the Discriminator into classifying its generated sample as true while the Discriminator is hoping to be correct in classifying the ground truth as real. This is an example of unsupervised learning. A popular use of this is to generate real-life images [29] or to transfer the *style* of one image and apply it to the content of another image [18]

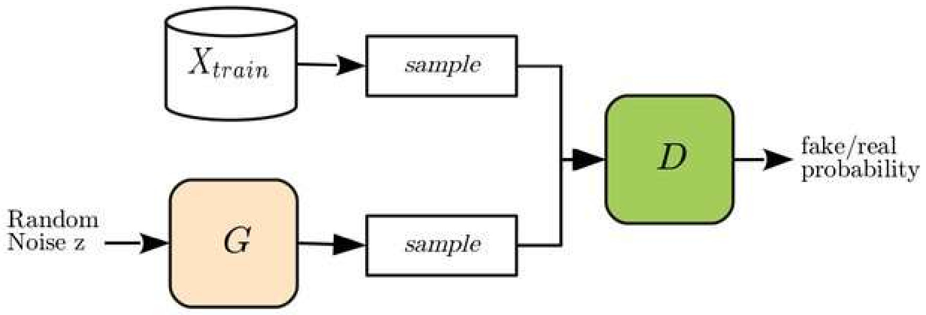

Figure 1.7: GAN (Picture by Emiliano De Cristofaro)

$$\min_G \max_D E_{x \sim q_{data}(x)}[log(D(x))] + E_{z \sim p(z)}[1 - D(G(z))]$$

The Discriminator wants to maximize this loss function meaning that D(x) must give a large value and D(G(z)) must give a small value (so the training sample should give a large value while the fake sample must give a small value). As outlined in the paper by Ian Goodfellow, we start by training the Discriminator and keeping the weights of the Generator fixed. Discriminator weights are updated using gradient ascent. After some iterations, the weights of the Discriminator are frozen and the weights of the Generator are updated using gradient descent.

[45] takes the idea of prediction a step further by predicting semantic segmentations instead of the raw image itself. A semantically labeled image consists of pixel-level labels where, instead of a bounding box, pixels belonging to certain classes receive the same label. The authors use a multiscale X2X model with two spatial scales followed by adversarial training. Adversarial training was combined with RNNs in [69]. An encoder takes a time-series mocap data as input and a novel geodesic loss with respect to the ground truth is used to train the encoder-decoder RNN structure. Two discriminators are used to make the predictions more continuous and real. The Fidelity Discriminator is trained to predict whether the predicted sequence is possible or not. And the Continuity Discriminator is applied on the entire sequence (input and predicted sequence) to check whether the sequence is continuous or not. [40] uses Flownet [17] to predict the optical flow for a sequence of images and a separate U-Net [51] is used to predict the future sequence. An optical flow loss, intensity loss and gradient loss are computed before finally using a Generator to check if the predicted sequence is real or not. If the GAN classifies it as unreal, then it is marked as an anomalous event.

[37] uses a VAE to encode and decode mean and variance values from historical trajectories. A decision vector is sampled from the latent vector and is used to aid an encoder by adding to its hidden state (the encoder also received the original historical trajectories as input). The new hidden state is then decoded before being sent to a discriminator which is trained to decide whether the forecasted paths are real or not.

1.2.4 Reinforcement Learning

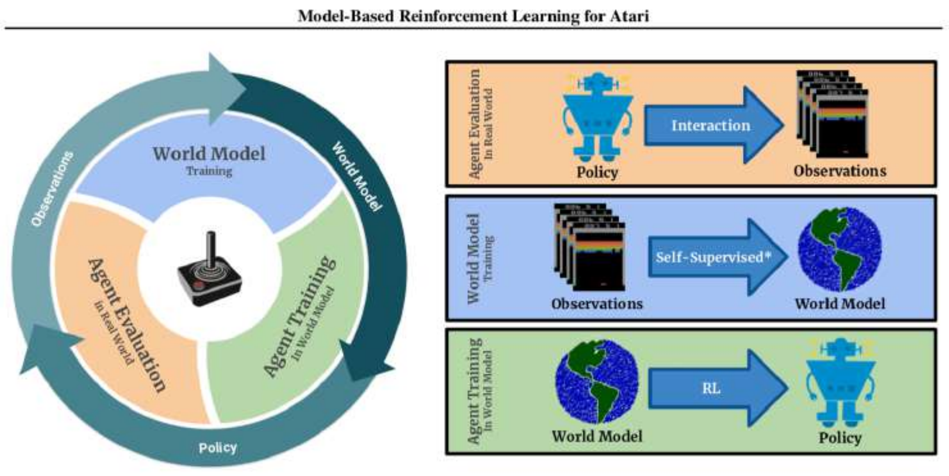

Figure 1.8: RL Agent plays Atari games (Picture by Ahmet Salim Bilgin)

Reinforcement learning algorithms have also proved to be useful for the task of trajectory prediction. The technique gained popularity in recent years when it was used to play Atari games [44]. An RL agent learns the best control policy using Q-Learning aided with Action Replay. A preprocessing step is necessary for dimensionality reduction. Convolution is used to extract features from a sequence of images which are finally fed to fully-connected layers before outputting an action that the agent is supposed to take in-game. In general, a reinforcement learning environment has an agent that takes certain actions to maximize a reward function while going from one state to the next. Doing this in a supervised fashion (an expert provides sample actions to take for given states) would be non-ideal since a lot of examples would need to be taken and, in theory, the agent might never be better than the expert. the alternative is to start with a random policy and try to discover the best one using policy gradients. Instead of outputting the action value, the network could provide probabilities associated to each action and have the agent choose its next action under the guise of exploitation (choose the action with the highest probability, the greedy approach) or exploration (choose a random action that may not look like the best action at that time but it can possibly pay off in the future). Another form of RL is inverse RL where the reward function is estimated, probabilities are computed using some optimal policy and then backprop is used to find a better reward function. An example of this can be found in [1]. Here, a car learns to master a driving simulator by trying to follow human examples as closely as possible.

[35] uses a very elaborate neural net to do trajectory prediction from an ego-centric and birdseye perspective [50]. The key idea behind their algorithm is to capture the multimodal nature underlying each prediction. A sample generation module creates the plausible trajectories using a combination of RNNs and CVAEs. These are refined using inverse reinforcement learning and a decoder network that uses a Scene Context Fusion cell to better capture the interaction between moving agents in a scene. [12] extracts relevant parts from an image which are then activated using reinforcement learning to decide what action will take place in the future.

1.3 Problems with previous approaches

The approaches mentioned so far are almost all supervised. This means that they require ground truth values to train the neural network. Even those neural networks that make multiple predictions, seem to almost always be biased towards certain values most commonly found in the dataset. For example, the KITTI dataset shows cars mostly driving forward and never making abrupt lane changes. Hence, a prediction algorithm trained on the dataset will output biased predictions. Another glaring issue with this approach is predicting the trajectory of stationary objects. In KITTI, Cityscapes, Berkley Deep Drive [78] and many other self driving datasets, stationary objects almost always never move. This means that any prediction algorithm trained on data like this will mostly predict static objects as remaining at their location. This may not be true, for example, a car in a parking place can decide to move out all of a sudden.

None of the previous approaches treat the problem of prediction, from an egocentric perspective, as a 3D problem. This is a major oversight because depth is an important factor since all real-life motion takes place in 3D and is simply projected onto a 2D image plane where the notion of depth is lost.

Chapter 2

Methodology

Before talking about the final proposed method, it is important to talk about the other methods that were used to get to the final approach. The drawbacks of these approaches can help us understand the numerous aspects of this problem. Then, as we build upon these numerous methods, we can eventually get to the final approach in a more systematic way.

2.1 2D Kalman Filter

As mentioned before, the problem of prediction from an ego-centric perspective is a 3D problem and not a 2D problem. This is because, what we see on the image plane is missing the notion of depth. To make a proper prediction, the 3D aspect of the problem needs to be taken into account. To demonstrate how a simple 2D approach can fail in certain scenarios, we look at some examples with a Kalman filter. The constant velocity model is used and the first 5 measurements are used to make a prediction 10 frames into the future. For a given object, a state vector is defined as $X = [x, \dot{x}, y, \dot{y}]$ where x, y represent the position in the image plane and $\dot{x}, \dot{y}$ represent rate of change of x and y in the image plane. Let measurement vector be $z = [z_x,\ z_y]$ and the dynamics defining the relation between z and X be:

$$z = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} X$$

Since the constant velocity model is being used, that means the measured values are updated according to the following dynamics (n represents the number of samples before the prediction process starts, in our case, $n = 5$):

$$x[t] = x[t-1] + dt \times \dot{x}[t-1]$$

$$\dot{x}[t-1] = \dot{x}[t]$$

$$y[t] = y[t-1] + dt \times \dot{y}[t-1]$$

$$\dot{y}[t-1] = \dot{y}[t]$$

$$\dot{x}[0] = \tfrac{x[n]-x[n-1]}{n-1}$$

$$\dot{y}[0] = \tfrac{y[n]-y[n-1]}{n-1}$$

In the below examples, the Kalman Filter does a pretty good job for straightforward situations like cars traveling straight down a road or simply taking a turn. However, the Kalman Filter only predicts in the direction of the vector along which the bounding box values are changing.

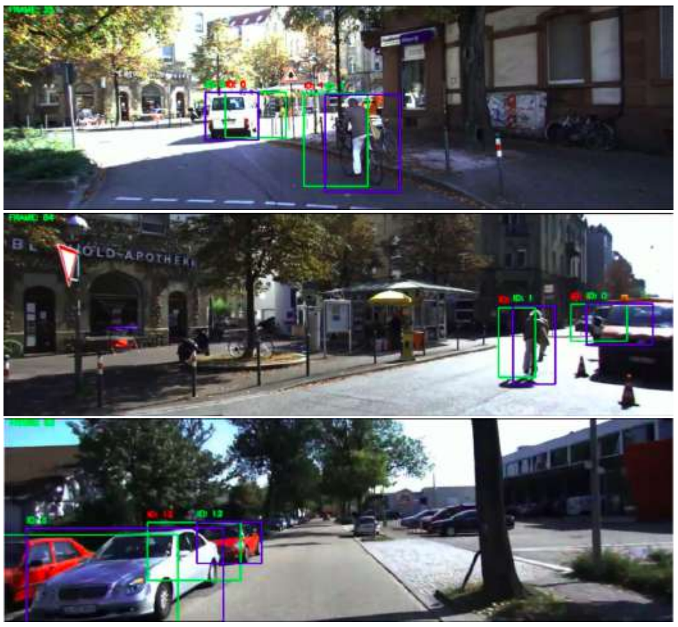

Figure 2.1: KITTI with Kalman Filter Example 1 (blue is the current ground truth bounding box and green is the Kalman filter prediction)

Some corner cases where the Kalman filter fails to predict in the right direction are given below.

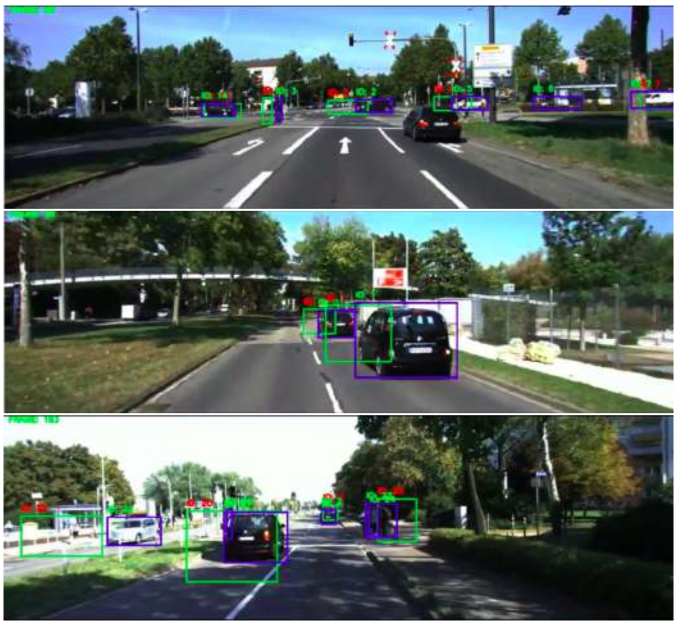

Figure 2.2: KITTI with Kalman Filter Example 2 (blue is the current ground truth bounding box and green is the Kalman filter prediction)

We see the issue here is that Kalman filter is unable to accurately predict the position of stationary objects. This is because, as the ego-car gets closer to the stationary object, its bounding box gets bigger. And so, because the Kalman filter only takes as input the 2D bounding box coordinates, it predicts the future coordinates based on that. Unsurprisingly, for a stationary object, the predicted bounding box coordinates represent a bounding box that is larger than the actual one because as we get closer to the stationary object, its bounding box gets bigger and is displaced in either direction based on how the ego-car approaches the object. This direction of displacement may or may not be the correct one. In the middle picture, we see another issue that arises due to the Kalman filter's failure to predict multiple future coordinates. We see that the car ahead of us has a prediction along its current lane. While that may be reasonable, another possible situation could arise if the car decides to change lanes in an effort to overtake the vehicle in front of it. However, we see clearly that the Kalman filter is unable to do that since the only information it receives are the bounding box coordinates. To make better predictions, we need to utilize the whole 3D space and also image information.

2.2 3D Bounding Box Propagation

In this approach, 3D bounding boxes are combined with semantic information to predict the future likely location of an object. The assumption here is that an object like a car, cyclist or person will always continue in the direction they are facing.The algorithm is summarized as follows:

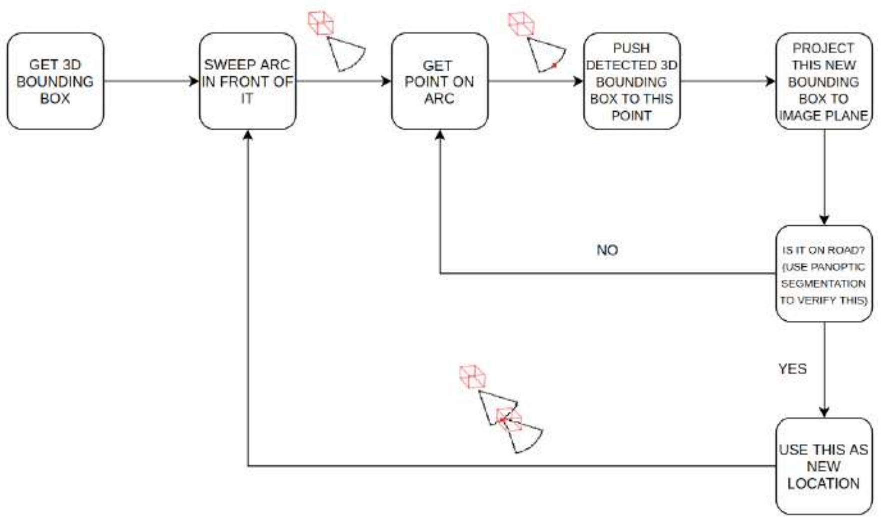
GET 3D BOUNDING BOX
SWEEP ARC IN FRONT OF IT
GET POINT ON ARC
PUSH DETECTED 3D BOUNDING BOX TO THIS POINT
PROJECT THIS NEW BOUNDING BOX TO IMAGE PLANE
IS IT ON ROAD? (USE PANOPTIC SEGMENTATION TO VERIFY THIS)
NO
YES
USE THIS AS NEW LOCATION

Figure 2.3: 3D Bounding Box Propagation

First the 3D bounding box for the object is obtained. Given the point cloud (obtained via laser scanner), 3D object detector is applied to it to get the 3D bounding boxes (length, width, height, heading angle, and location of the bounding box centroid). At this stage, many object detectors were experimented with and their results on the KITTI dataset are given below (from the KITTI 3D object detection leaderboard):

Name	inference time (s/im)	box AP car (hard)	box AP pedestrian (hard)	box AP cyclist (hard)
MMLab PV-RCNN [53] [55] [54]	0.08	38.6	40.29	57.65
3DSSD [75]	0.04	74.55	40.23	56.90
Point-GNN [56]	0.6	72.29	40.14	57.08
F-ConvNet [70]	0.47	66.69	38.80	56.54
Point-Pillars [12]	**0.016**	**68.99**	**38.89**	**51.92**
AVOD-FPN [33]	0.1	65.73	39.04	44.93
F-PointNet [48]	0.17	60.59	38.08	49.01

The evaluation metric used for comparison is the box AP (average precision). For the hard AP, true positive is one which has an IoU, with the ground truth box, of at least 50% for pedestrians and cyclists and at least 70% for vehicles. PointPillars was ultimately used due to its reasonable compromise on speed and accuracy.

Another way to obtain the point cloud data was directly from stereo images. Here, the Pseudo-LIDAR approach is used where the disparity map is converted to a depth map, which is then converted to a point cloud using the following formula:

$$(depth) \quad z = \frac{f_U \times b}{Y(u,v)}$$
$$(width) \quad x = \frac{(u - c_U) \times z}{f_U}$$
$$(height) \quad y = \frac{(v - c_V) \times z}{f_V}$$

Where *Y(u,v)* represents the intensity of the depth map at pixel value (u,v). f_U, f_V, c_U c_V and b represent the horizontal and vertical focal lengths, pixel locations corresponding to the camera center and the baseline respectively.

Three neural nets were primarily experimented with and MADNet was ultimately selected for its reasonable compromise on speed and accuracy. First PSMNet was tried [11] where two weight-sharing CNNs were used for feature map calculation. An SPP (Spatial Pyramid Pooling) module was used for feature harvesting by concatenating representations from subregions with different sizes, and a convolution layer for feature fusion. Output of this network is a depth map where the value of each (x, y) coordinate is the distance from the camera. GANet [80] used semiglobal guided aggregation (SGA) and local guided aggregation (LGA) layers. The authors designed a new semi-global cost aggregation step which supports backpropagation. This is more effective than the traditional SGM and can be used repetitively in a deep neural network model to boost the cost aggregation effects. On the other hand, the local guided aggregation (LGA) layer refines the thin structures and object edges. Real-Time Self-Adaptive Deep Stereo [62] deals with domain shift by doing partial (MAD) or full backprop during online operation. This means that the network is always in training mode as it fine tunes itself according to the unseen data. Comparative results for all three on the KITTI dataset are given below (taken from the KITTI Stereo Evaluation 2015 leaderboard). One thing to notice here is that MADnet is significantly faster than the other two with a reasonable drop in performance.

Name	D1-bg	D1-fg	D1-all	Runtime
GANet-deep	1.48%	3.46%	1.81%	1.8s
PSMNet	1.86%	4.62%	2.32%	0.41s
MADnet	**3.75%**	**9.20%**	**4.66%**	**0.02s**

In the vehicle coordinate frame, an arc of 60 degrees is generated in front of the 3D object bounding box based on its rotation angle (yaw angle). 60 degrees makes sense because most vehicles can steer upto 30 degrees in both directions. Radius of the arc was kept at 1 meters. 10 points are picked on the boundary of the arc such that they are equidistant from each other. First, the farthest point is chosen as a likely future location for the 3D bounding box. The bounding box is redrawn on this point with the same dimensions as before but with the rotation recalculated according to: $\theta_{new} = \theta_{original-yaw} + \theta_{new-yaw}$. It is then projected back to the image plane. The base of the bounding box must have an IoU of 90% with the road/ pavement semantic labels. For semantic labeling of road, pavement and grass, we use neural nets that provide panoptic segmentation. If the IoU is less, then the next farthest point is chosen and the process is repeated until a likely future location for the bounding box is discovered. Panoptic segmentation is a combination of instance and semantic segmentation. In the former, specific instances of the object are separately labeled pixel-wise. So, for example, you have multiple cars and each of those cars are detected via 2D bounding box and then every pixel belonging to an instance of a car is labeled as car 1 and the same is applied to all other detected cars (their pixels are labeled as car 2, car 3 etc.). In semantic segmentation, we don't discriminate between the different detected instances of a class. So, in the previous example, all pixels belonging to an object labeled as car would be classified simply as car and not as car 1, car 2 etc. In panoptic segmentation, you assign two labels to each pixel: The instance label (car 1) and the class label (car). This technique was used for generating a semantic map for the road, pavement, and grass in the image. Below are some of the neural nets that were experimented with (results compiled for the Cityscapes Dataset). These results were taken from the Cityscapes leaderboard for panoptic segmentation.

Name	PQ	PQST	PQTH	MIOU	AP
Panoptic DeepLab (X71) [13]	64.1	–	–	81.5	38.5
Panoptic FPN (ResNet 101) [34]	58.1	62.5	52	75.7	33
Mask R-CNN + COCO [2]	–	–	54.0	–	–
TRI-ML [26]	**58.8**	**63.7**	**52.1**	–	–

[26] was used to get the panoptic segmentation. The algorithm is recursive in the sense that the prediction process of sweeping an arc with respect to a position is repeated. Only this time, the verified position is used as the new center and the arc is drawn with respect to it. Ideally, the recursion would keep going on until a point comes where none of the chosen points on the boundary of the arc are valid predictions (they have a low IoU with the road/ pavement semantic labels, i.e. they are not completely on those surfaces when projected back to the image plane). For cars, only IoU with

road pixels is computed. For cyclists, IoU with both road and pavement pixels is computed. And, for pedestrians, the IoU is computed with both road, pavement, and grass pixels. Below, we can see that this algorithm can predict 3D bounding boxes reasonably far into the future.

This algorithm is obviously much better than the Kalman Filter approach. It takes into account the 3D aspect of the problem and also uses information from the image to make better predictions.

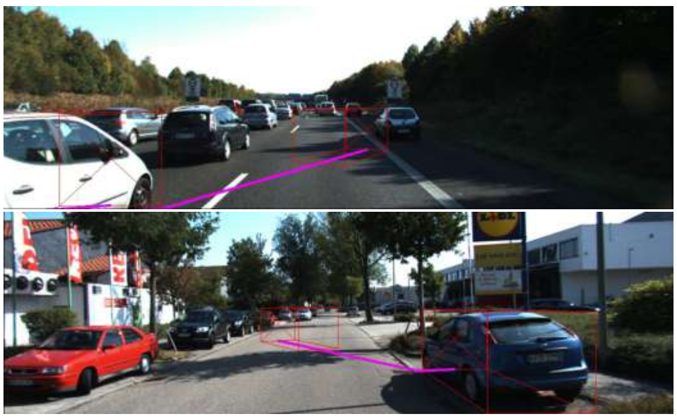

Figure 2.4: KITTI with 3D bounding box propagation upto a single level of recursion (pink is the current bounding box and yellow is the prediction)

Before, stationary objects were predicted to remain in the same position. This is because the Kalman filter would look at the first five position values and since they were more or less the same, the predicted future values wouldn't be that different from the most recent bounding box values. This is not the case here because we use information from the image and we also assume that objects that are currently stationary might try to move in the future and, the additional assumption, that they might try to move in the direction they are currently facing.

This approach is good but it has two drawbacks. It is computationally expensive as you recursively try to predict the bounding box as far as possible (or as far as you wish) until the road can no longer be observed in the image plane. One can also use all the points on the boundary of the arc and recursively predict the future bounding box in this way and obtain many (multiple) future locations. However, this would not be feasible if real-time operation is the main focus. A much better approach would be to use a more continuous representation for the predictions instead of a discretized approach with several 3D bounding boxes represented at predefined distances from the current position of the object.

2.3 Potential Field

As in the previous approach, we use the current heading of the 3D bounding box to obtain a future proposal by simply pushing the bounding box forward in space by a few meters from its current position. After that, this newly proposed bounding box is projected to the image plane using the camera matrix and its validity is ascertained by computing the IoU of its base with the road, pavement, and/ or grass pixels (semantically labeled by a neural net). This means, that the proposed future bounding box location is actually a good predicted position for the current object. Multiple proposals are obtained in this way. Below, is a visual representation of this process:

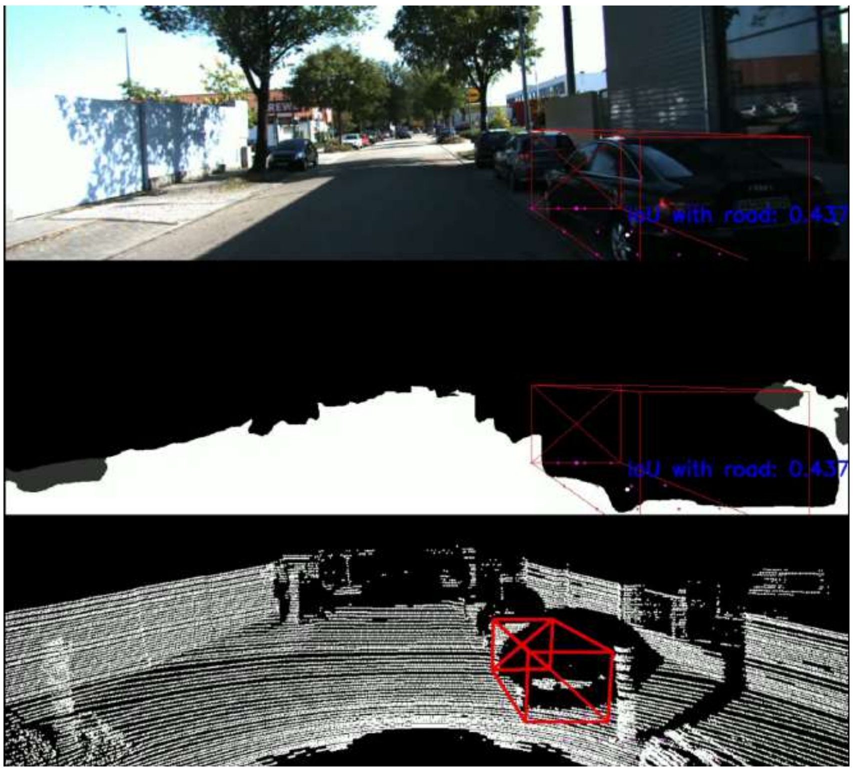

Figure 2.5: top most picture shows the 3D bounding box in the original image, middle picture shows the bounding box in the image plane with only the semantically labeled pixels shown (for road and pavement), and the bottom picture shows the same situation from a slightly elevated perspective with only point cloud present

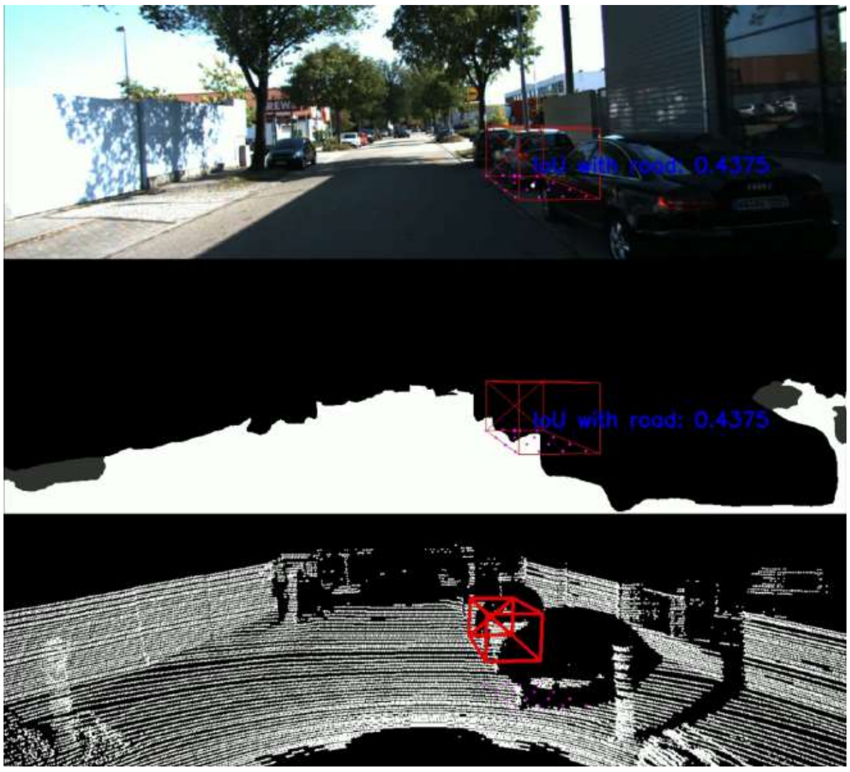

Figure 2.6: Now the bounding box is pushed forward in space upto a maximum limit and all valid positions are selected based on the IoU of the base of the projected 3D bounding box (the one being pushed forward from its original position) with the road pixels (which were semantically labeled with the help of a neural net). We can see that the proposed bounding box has a very low IoU because there are other cars parked in front

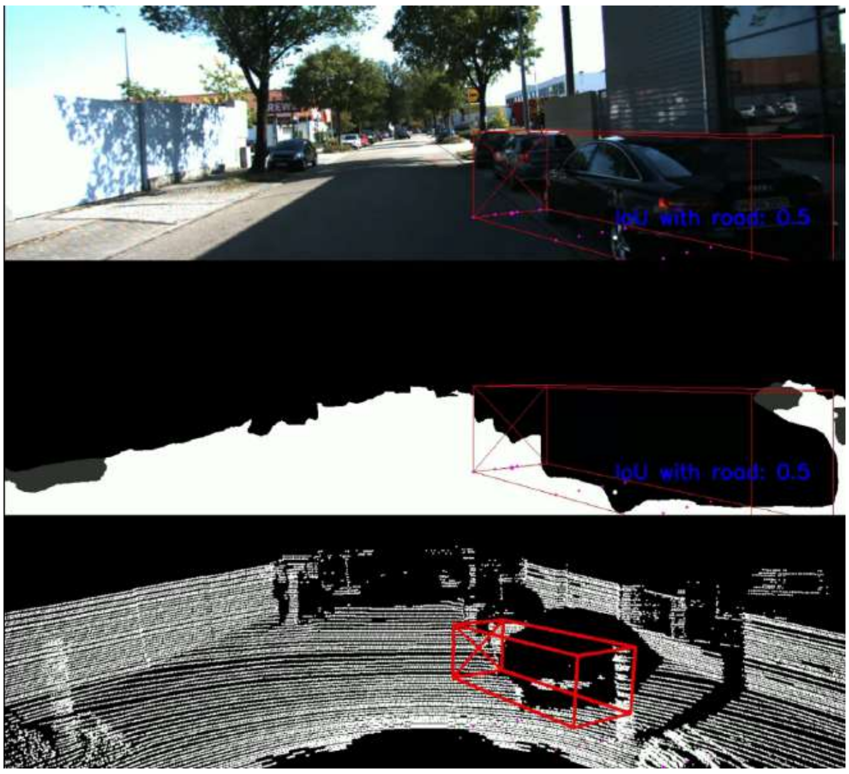

Figure 2.7: In order to look for better proposals, the bounding box is rotated in the anticlockwise direction by 10 degrees (it is rotated upto a prescribed maximum angle in the anticlockwise direction from its current heading angle and then rotated upto the same prescribed maximum angle in the clockwise direction)

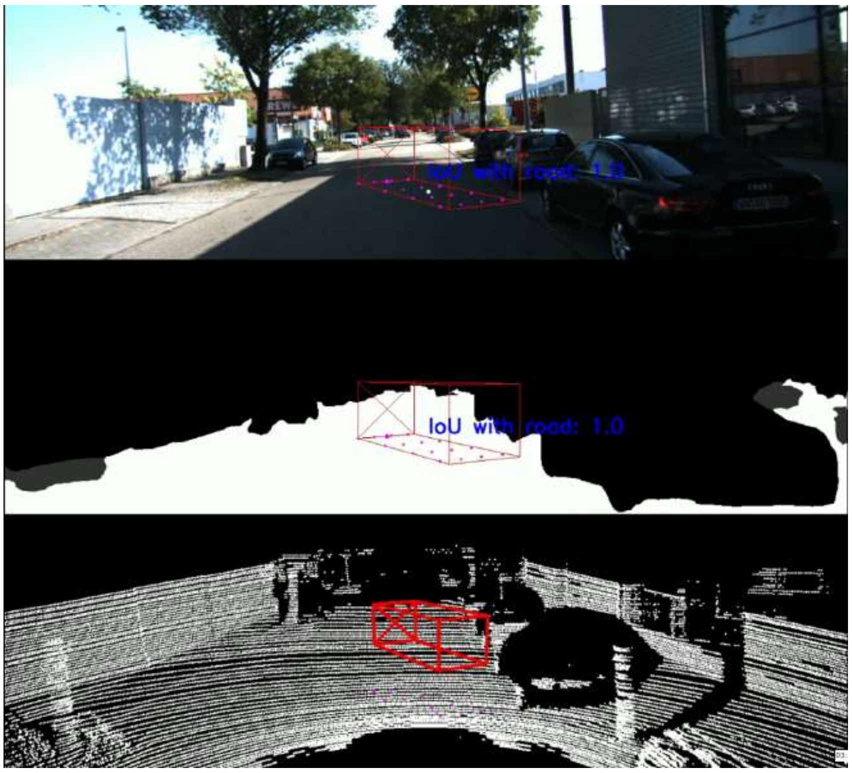

Figure 2.8: Now, better proposals are obtained since the new forward-projected bounding box has a really high IoU with the road

Instead of displaying several bounding boxes, we can take the center of the base of these proposals and use them to generate a potential field in front of the object. So, imagine all future proposals as points within a cone in front of the detected object and each point is the center of the base of a 3D bounding box that was pushed forward in space from the current position of the detected bounding box.

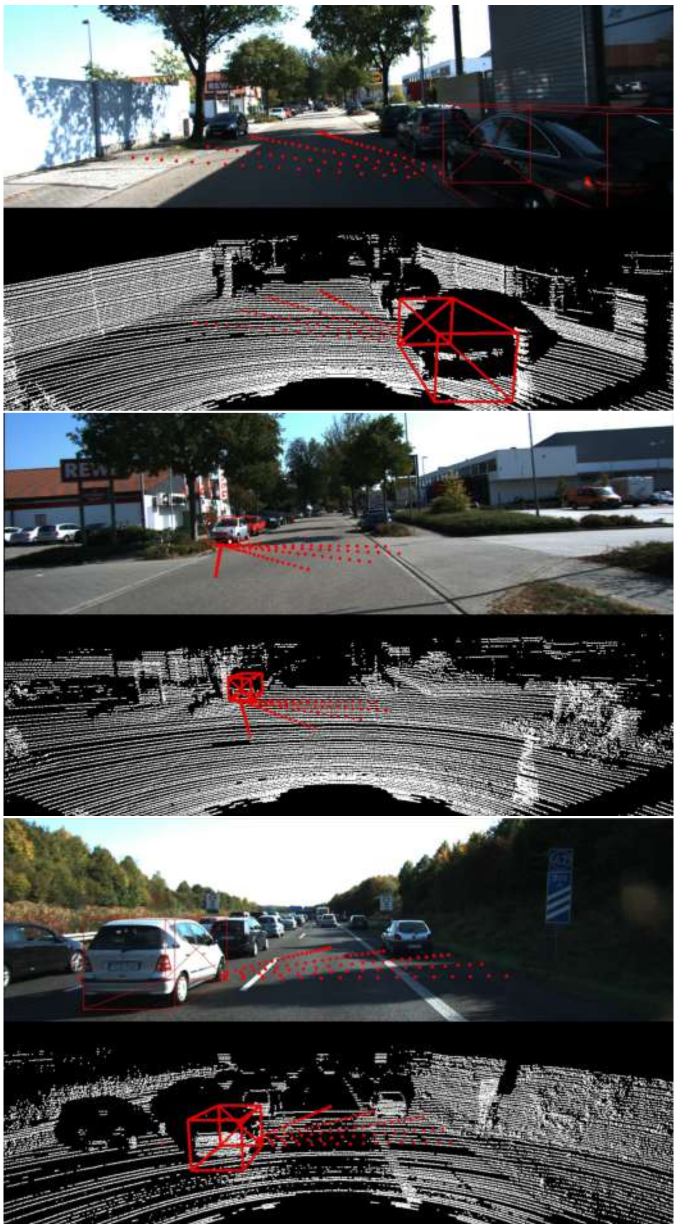

Figure 2.9: A radial potential field growing outward (for each top-bottom pair, the bottom picture represents the potential field in the point cloud domain and the top picture represents the same in the image plane)

The previous recursive approach is abandoned in favor of a continuous expansion from the current position of the detected object upto a limit. This is also much easier to understand as compared to plotting numerous bounding boxes. However, despite being less discretized, this approach can still be made better and more continuous and, also, more visually appealing.

2.4 Heatmap

In this section, the method that is ultimately used to do trajectory prediction is presented. We started with simple 2D bounding box proposals and then moved into the 3D domain where we formalized a method that combines 3D bounding boxes with the panoptic segmentation to get future 3D bounding boxes for a detected object. Then we simplify that process and obtain a radial potential field. And now, we convert that radial potential field into a heatmap that is 'hot' close to the object and gets 'warmer' and eventually 'cold' at a certain limit from the object. To obtain this heatmap, we use the same approach for getting the potential field but now a larger number of points are used.

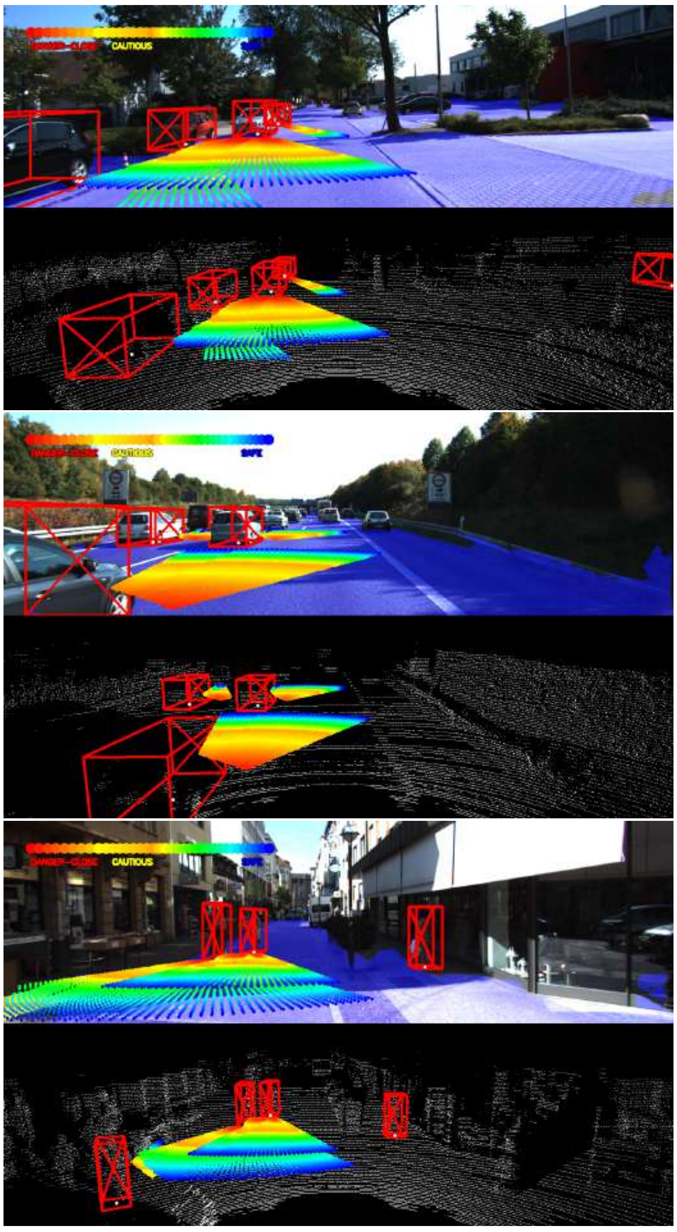

Figure 2.10: A heatmap that grows outwards in the form of a 60 degree cone, it is hot close to the object but gets colder as you move far away. Clearly, this is visually appealing and more informative than all the previously mentioned approaches

We can go a step further and utilize the heatmap to predict the future motion of the object. Specifically, we can predict whether an object would turn left, turn right or just go straight. To do this, we divide the cone into three parts (left, right, and forward) and the size of the heatmap (determined by the number of points) in each part is recorded and then the probability is computed as the size of the heatmap in a given part divided by the total size of the entire heatmap.

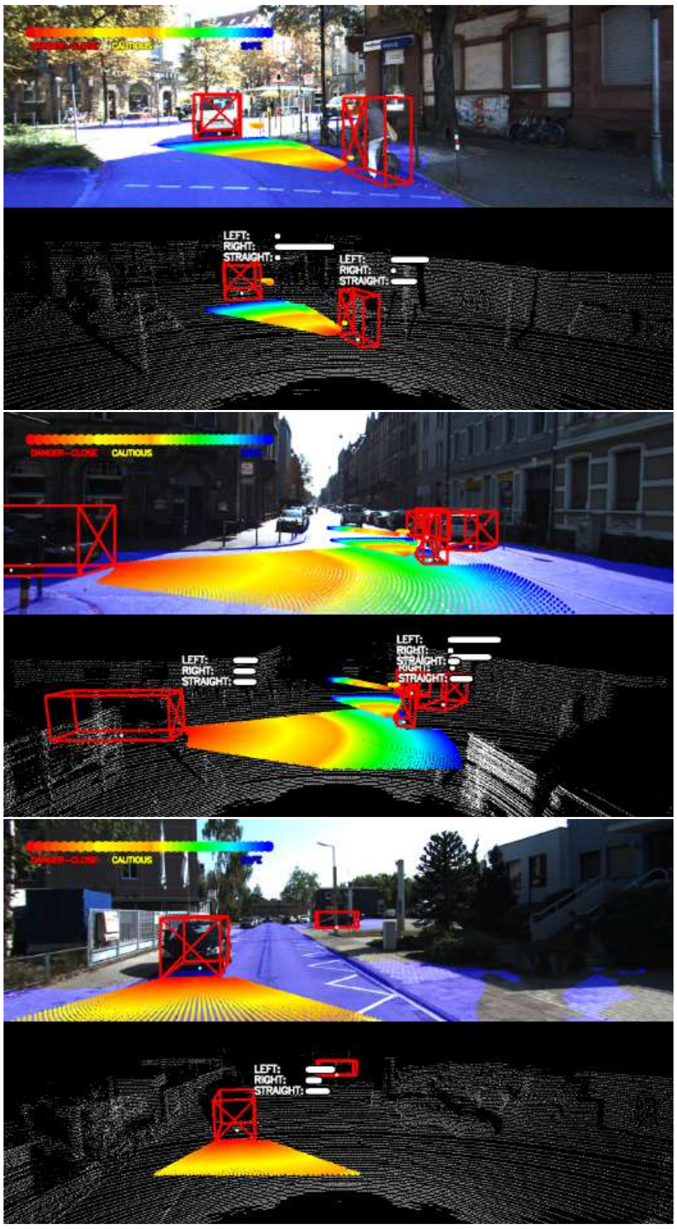

Figure 2.11: Probability that an object might turn left, turn right, or go straight (calculated using the heatmap)

2.5 Speed and Steering Value Recommendation System Prototype

As a way to show the potential usefulness of the heatmap approach, a simple recommendation system was developed. This would provide the speed and steering recommendation based on the heatmap. Since actual speed and steering values were not provided for the KITTI dataset, some assumptions on the default values were made. Specifically, it is assumed that the car, by default, always goes straight and would always move with the maximum possible speed. To make speed and steering recommendation, an arc is generated in front of the ego-car and its intersection with the heatmap is calculated. If, for example, the arc has a greater intersection with detected objects and their heatmaps towards the left, as compared to the right side, then that means there is heavy traffic in that direction. And so the recommended steering is towards the right. The speed recommendation is based on the number of times the arc intersects with 3D bounding boxes and their heatmaps.

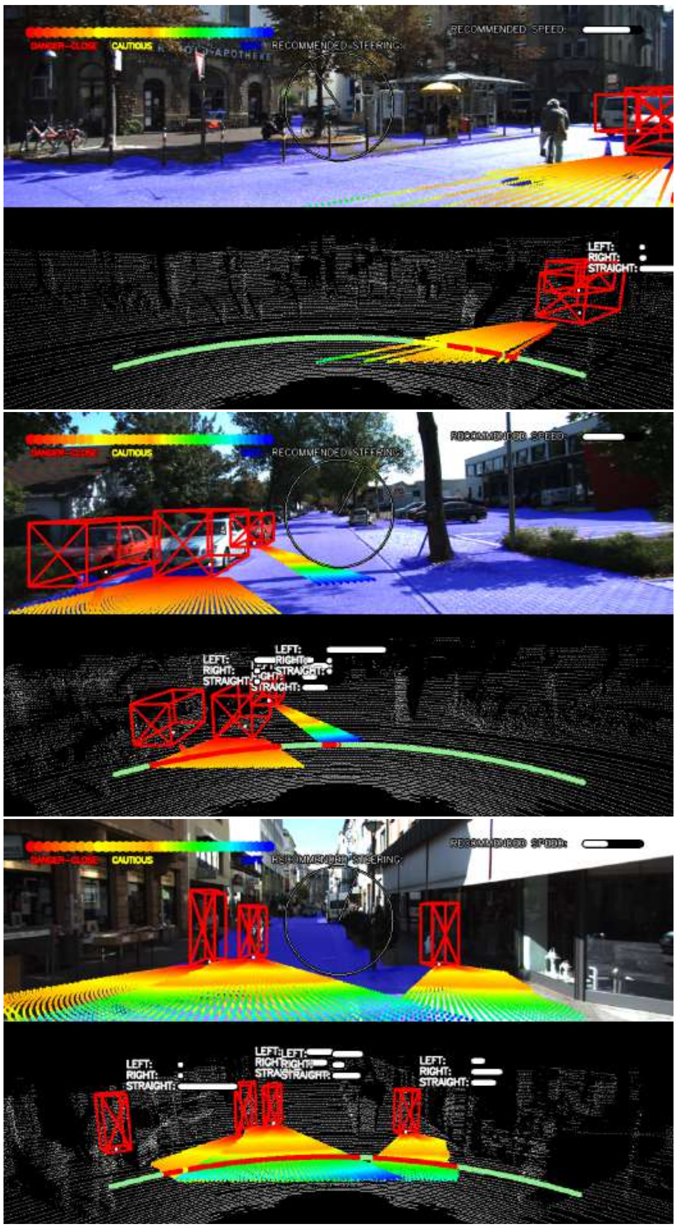

Figure 2.12: Speed and steering recommendation based on 3D bounding box and its heatmap

Chapter 3

Results

Experiments were performed with different stereo depth generating neural nets. All networks were trained on Google Colab (Tesla K80 GPU and 25 GB of RAM). MADNet (full backprop, partial backprop, and no backprop), GANet, and PSMNet were used on the KITTI object detection dataset. Pretrained neural nets were available for all three and were used to generate the depth map for all the images in the dataset. Then, Pseudo-LIDAR technique was used to convert the depth map to a point cloud and this was made sparse [77] to decrease the size of the input to the 3D object detector (PointPillars will process a less dense cloud much quicker than a denser one with more 3D points). Below are the results in which PointPillars was trained and used for 3D object detection on fake laser scanner data generated using the Pseudo-LIDAR approach:

Name	inference time (s/im)	box AP car (hard)	box AP pedestrian (hard)	box AP cyclist (hard)
GANet	1.52	59.08	25.6	40.29
PSMNet	0.41	55.6	24.03	40.01
MADNet (full backprop)	0.089	50.01	22.01	32.5
MADNet (partial backprop)	0.03	48.02	20.19	30.1
MADNet (no backprop)	0.01	44.1	18.13	26.04

Despite the decrease in accuracy over using actual laser scanner data, because the panoptic segmentation is ultimately used to decide whether a proposal for predicted future position is valid or not, an inaccurate bounding box doesn't create that many problems. An example is given below:

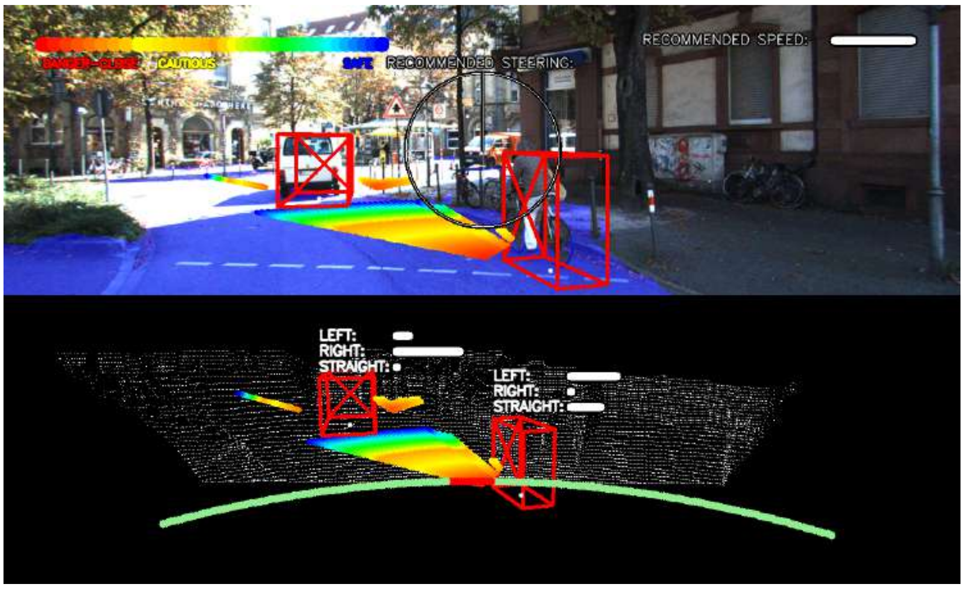

Figure 3.1: Heatmap of inaccurate 3D bounding box is guided by the panoptic segmentation

However, there were some situations in which the slight inaccuracy of the bounding box created problems. This usually happened on two-way narrow roads where both sides were separated by a barrier:

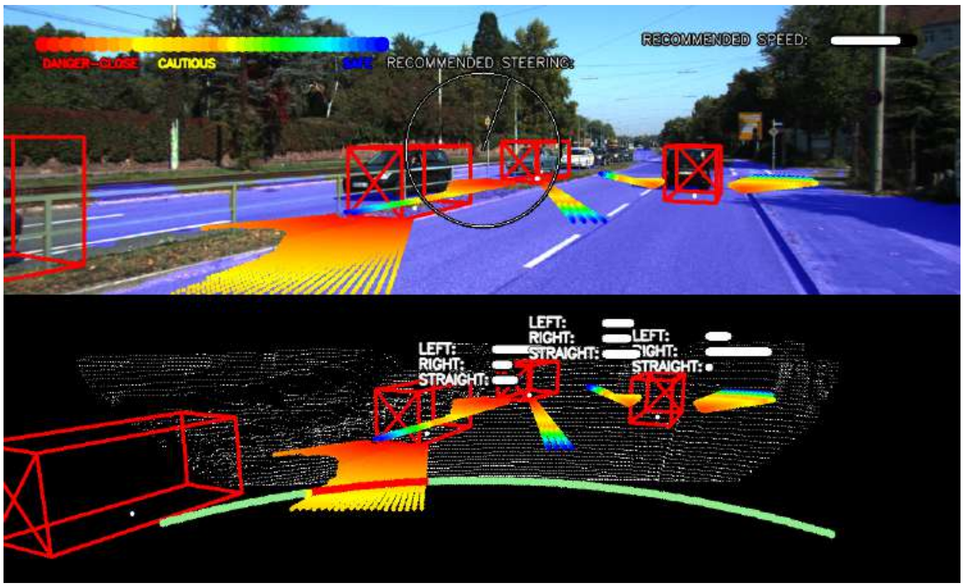

Figure 3.2: Heatmap of inaccurate 3D bounding box is in the wrong lane

There were also cases where the panoptic segmentation didn't perform so well. This happened in the Waymo Open [60] Dataset, specifically during the night time scenarios. However, this problem can easily be alleviated by finding a better neural network to do panoptic segmentation.

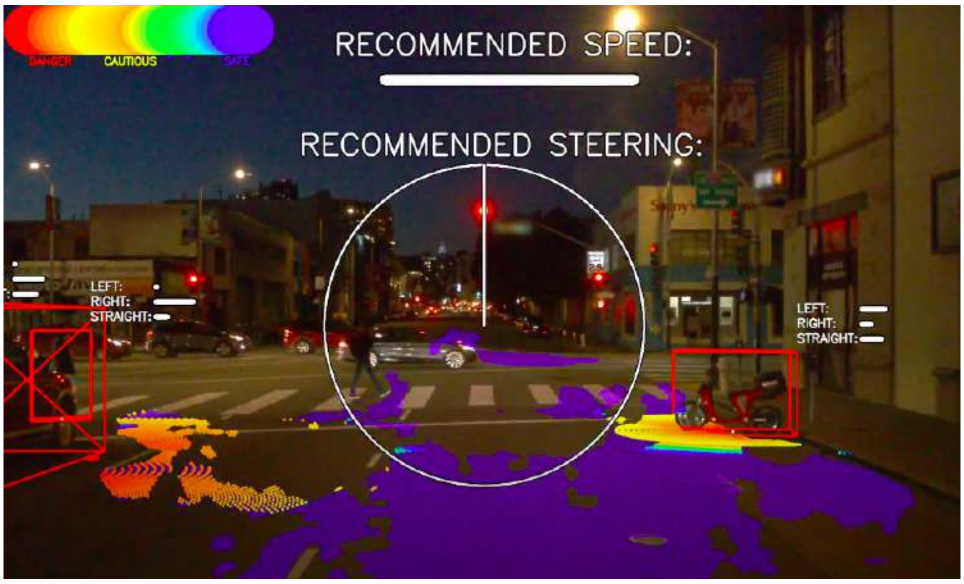

Figure 3.3: Inaccurate panoptic segmentation, at night-time, has an effect on the heatmap generation

Due to the use of real-time neural nets, the solution has a total time of 0.3s. Below is a block diagram that represents the entire pipeline:

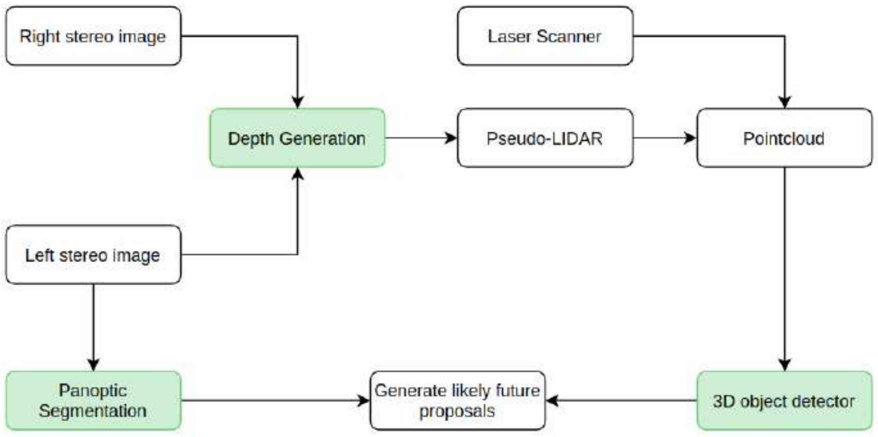

Figure 3.4: Block diagram representing the entire process

The pipeline is robust because, instead of end-to-end learning, separate problems are solved and their solutions are combined to get the final desired solution. In this case, the separate problems are 3D object detection and panoptic segmentation and also depth generation (if only stereo images are to be used). If a neural net is available for any of the mentioned tasks, we can simply replace the current component in the pipeline with a new one. So, for example, if a better 3D object detector is available, we can replace our current detector in the pipeline with this new one. All previous solutions try to do end-to-end learning and that suffers from the same kind of problems that plague supervised learning. No such problem is encountered here.

Conclusions

In this research, a new method for trajectory prediction was presented which can work with or without a LIDAR. A heatmap of possible future positions is then generated and this is used to calculate the probability of an object going left, right, or straight. A small prototype recommendation system was also presented that provides a speed and steering value based on the detected objects and their predicted trajectory. Future work and the obvious sequel to this research is to experiment with different self-driving car algorithms and see how a participating agent can use such a heatmap to make better driving decisions.

Bibliography

[1] Pieter Abbeel and Andrew Y. Ng. Apprenticeship learning via inverse reinforcement learning. In *In Proceedings of the Twenty-first International Conference on Machine Learning*. ACM Press, 2004.

[2] Waleed Abdulla. Mask r-cnn for object detection and instance segmentation on keras and tensorflow. `https://github.com/matterport/Mask_RCNN`, 2017.

[3] A. Alahi, V. Ramanathan, and L. Fei-Fei. Socially-aware large-scale crowd forecasting. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2211–2218, 2014.

[4] 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.

[5] Gianluca Antonini, Michel Bierlaire, and Mats Weber. Discrete choice models for pedestrian walking behavior. *Transportation Research Part B: Methodological*, 40:667–687, 09 2006.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate, 2014.

[7] Alex Bewley, ZongYuan Ge, Lionel Ott, Fabio Ramos, and Ben Upcroft. Simple online and realtime tracking. *CoRR*, abs/1602.00763, 2016.

[8] Apratim Bhattacharyya, Mario Fritz, and Bernt Schiele. Long-term onboard prediction of people in traffic scenes under uncertainty. *CoRR*, abs/1711.09026, 2017.

[9] Judith Bütepage, Michael J. Black, Danica Kragic, and Hedvig Kjellström. Deep representation learning for human motion prediction and classification. *CoRR*, abs/1702.07486, 2017.

[10] B. Cancela, A. Iglesias, M. Ortega, and M. G. Penedo. Unsupervised trajectory modelling using temporal information via minimal paths.

[11] Jia-Ren Chang and Yong-Sheng Chen. Pyramid stereo matching network. *CoRR*, abs/1803.08669, 2018.

[12] Lei Chen, Jiwen Lu, Zhanjie Song, and Jie Zhou. Part-activated deep reinforcement learning for action prediction. In *ECCV (3)*, pages 435–451, 2018.

[13] Bowen Cheng, Maxwell D Collins, Yukun Zhu, Ting Liu, Thomas S Huang, Hartwig Adam, and Liang-Chieh Chen. Panoptic-deeplab: A simple, strong, and fast baseline for bottom-up panoptic segmentation. In *CVPR*, 2020.

[14] Chung-Cheng Chiu, Tara N. Sainath, Yonghui Wu, Rohit Prabhavalkar, Patrick Nguyen, Zhifeng Chen, Anjuli Kannan, Ron J. Weiss, Kanishka Rao, Katya Gonina, Navdeep Jaitly, Bo Li, Jan Chorowski, and Michiel Bacchiani. State-of-the-art speech recognition with sequence-to-sequence models. *CoRR*, abs/1712.01769, 2017.

[15] 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.

[16] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. *CoRR*, abs/1604.01685, 2016.

[17] Philipp Fischer, Alexey Dosovitskiy, Eddy Ilg, Philip Häusser, Caner Hazirbas, Vladimir Golkov, Patrick van der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. *CoRR*, abs/1504.06852, 2015.

[18] Leon A. Gatys, Alexander S. Ecker, and Matthias Bethge. A neural algorithm of artistic style. *CoRR*, abs/1508.06576, 2015.

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

[20] Shaona Ghosh and Per Ola Kristensson. Neural networks for text correction and completion in keyboard decoding. *CoRR*, abs/1709.06429, 2017.

[21] Ian J. Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial networks, 2014.

[22] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013.

[23] Jiaxian Guo, Sidi Lu, Han Cai, Weinan Zhang, Yong Yu, and Jun Wang. Long text generation via adversarial training with leaked information. *CoRR*, abs/1709.08624, 2017.

[24] Tengda Han, Weidi Xie, and Andrew Zisserman. Video representation learning by dense predictive coding, 2019.

[25] Dirk Helbing and PÃ©ter MolnÃ¡r. Social force model for pedestrian dynamics. *Physical Review E*, 51(5):4282â4286, May 1995.

[26] Rui Hou, Jie Li, Arjun Bhargava, Allan Raventos, Vitor Guizilini, Chao Fang, Jerome Lynch, and Adrien Gaidon. Real-time panoptic segmentation from dense detections. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[27] Ashesh Jain, Hema Swetha Koppula, Shane Soh, Bharad Raghavan, Avi Singh, and Ashutosh Saxena. Brain4cars: Car that knows before you do via sensory-fusion deep learning architecture. *CoRR*, abs/1601.00740, 2016.

[28] Lukasz Kaiser and Samy Bengio. Can active memory replace attention? *CoRR*, abs/1610.08613, 2016.

[29] Tero Karras, Timo Aila, Samuli Laine, and Jaakko Lehtinen. Progressive growing of gans for improved quality, stability, and variation. *CoRR*, abs/1710.10196, 2017.

[30] Diederik P Kingma and Max Welling. Auto-encoding variational bayes, 2013.

[31] Kris M. Kitani, Brian D. Ziebart, James Andrew Bagnell, and Martial Hebert. Activity forecasting. In Andrew W. Fitzgibbon, Svetlana Lazebnik, Pietro Perona, Yoichi Sato, and Cordelia Schmid, editors, *ECCV (4)*, volume 7575 of *Lecture Notes in Computer Science*, pages 201–214. Springer, 2012.

[32] Julian Francisco Pieter Kooij, Nicolas Schneider, Fabian Flohr, and Dariu M. Gavrila. Context-based pedestrian path prediction. In David J. Fleet, TomÃ¡s Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *ECCV (6)*, volume 8694 of *Lecture Notes in Computer Science*, pages 618–633. Springer, 2014.

[33] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven Waslander. Joint 3d proposal generation and object detection from view aggregation. *IROS*, 2018.

[34] Justin Lazarow, Kwonjoon Lee, Kunyu Shi, and Zhuowen Tu. Learning instance occlusion for panoptic segmentation. In *The IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

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

[36] W. Li, V. Mahadevan, and N. Vasconcelos. Anomaly detection and localization in crowded scenes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(1):18–32, 2014.

[37] Yuke Li. Which way are you going? imitative decision learning for path forecasting in dynamic scenes. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[38] Junwei Liang, Lu Jiang, Juan Carlos Niebles, Alexander G. Hauptmann, and Li Fei-Fei. Peeking into the future: Predicting future person activities and locations in videos. *CoRR*, abs/1902.03748, 2019.

[39] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. Recurrent neural network for text classification with multi-task learning. *CoRR*, abs/1605.05101, 2016.

[40] Wen Liu, Weixin Luo, Dongze Lian, and Shenghua Gao. Future frame prediction for anomaly detection - A new baseline. *CoRR*, abs/1712.09867, 2017.

[41] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015.

[42] Julieta Martinez, Michael J. Black, and Javier Romero. On human motion prediction using recurrent neural networks. *CoRR*, abs/1705.02445, 2017.

[43] Ramin Mehran, Alexis Oyama, and Mubarak Shah. Abnormal crowd behavior detection using social force model. In *CVPR*, pages 935–942. IEEE Computer Society, 2009.

[44] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[45] Natalia Neverova, Pauline Luc, Camille Couprie, Jakob J. Verbeek, and Yann LeCun. Predicting deeper into the future of semantic segmentation. *CoRR*, abs/1703.07684, 2017.

[46] Mingtao Pei, Yunde Jia, and Song Zhu. Parsing video events with goal inference and intent prediction. pages 487–494, 11 2011.

[47] Stefano Pellegrini, Andreas Ess, Konrad Schindler, and Luc Van Gool. You'll never walk alone: Modeling social behavior for multi-target tracking. In *ICCV*, pages 261–268. IEEE Computer Society, 2009.

[48] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. Frustum pointnets for 3d object detection from rgb-d data. *arXiv preprint arXiv:1711.08488*, 2017.

[49] 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.

[50] A. Robicquet, A. Sadeghian, A. Alahi, and S. Savarese. Stanford drone dataset.

[51] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. *CoRR*, abs/1505.04597, 2015.

[52] Mike Schuster and Kuldip K. Paliwal. Bidirectional recurrent neural networks. *IEEE Trans. Signal Process.*, 45(11):2673–2681, 1997.

[53] Shaoshuai Shi, Chaoxu Guo, Li Jiang, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538, 2020.

[54] Shaoshuai Shi, Xiaogang Wang, and Hongsheng Li. Pointrcnn: 3d object progposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779, 2019.

[55] Shaoshuai Shi, Zhe Wang, Jianping Shi, Xiaogang Wang, and Hongsheng Li. From points to parts: 3d object detection from point cloud with part-aware and part-aggregation network. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2020.

[56] Weijing Shi and Ragunathan (Raj) Rajkumar. Point-gnn: Graph neural network for 3d object detection in a point cloud. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2020.

[57] Mennatullah Siam, Sepehr Valipour, Martin Jägersand, and Nilanjan Ray. Convolutional gated recurrent networks for video segmentation. *CoRR*, abs/1611.05435, 2016.

[58] Deepika Singh, Erinc Merdivan, Ismini Psychoula, Johannes Kropf, Sten Hanke, Matthieu Geist, and Andreas Holzinger. Human activity recognition using recurrent neural networks. *CoRR*, abs/1804.07144, 2018.

[59] Shashank Srikanth, Junaid Ahmed Ansari, Karnik Ram R., Sarthak Sharma, J. Krishna Murthy, and K. Madhava Krishna. INFER: intermediate representations for future prediction. *CoRR*, abs/1903.10641, 2019.

[60] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, Vijay Vasudevan, Wei Han, Jiquan Ngiam, Hang Zhao, Aleksei Timofeev, Scott Ettinger, Maxim Krivokon, Amy Gao, Aditya Joshi, Yu Zhang, Jonathon Shlens, Zhifeng Chen, and Dragomir Anguelov. Scalability in perception for autonomous driving: Waymo open dataset, 2019.

[61] Martin Sundermeyer, Ralf SchlÃ$\frac{1}{4}$ter, and Hermann Ney. Lstm neural networks for language modeling.

[62] Alessio Tonioni, Fabio Tosi, Matteo Poggi, Stefano Mattoccia, and Luigi Di Stefano. Real-time self-adaptive deep stereo. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[63] F Tung, J S. Zelek, and D A. Clausi. Goal-based trajectory analysis for unusual behaviour detection in intelligent surveillance. *Image and Vision Computing*, 29:230 – 240, 2011.

[64] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: Lessons learned from the 2015 MSCOCO image captioning challenge. *CoRR*, abs/1609.06647, 2016.

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

[66] Jacob Walker, Abhinav Gupta, and Martial Hebert. Patch to the future: Unsupervised visual prediction. In *CVPR*, pages 3302–3309. IEEE Computer Society, 2014.

[67] Xiaogang Wang, Xiaoxu Ma, and W.E.L. Grimson. Unsupervised activity perception in crowded and complicated scenes using hierarchical bayesian models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 31(3):539–555, mar 2009.

[68] Yan Wang, Wei-Lun Chao, Divyansh Garg, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar from visual depth estimation: Bridging the gap in 3d object detection for autonomous driving. *CoRR*, abs/1812.07179, 2018.

[69] Yuxiong Wang, Liang-Yan Gui, Xiaodan Liang, and Jose M. F. Moura. Adversarial geometry-aware human motion prediction. In *Proceedings of European Conference on Computer Vision (ECCV)*. Springer, October 2018.

[70] Zhixin Wang and Kui Jia. Frustum convnet: Sliding frustums to aggregate local point-wise features for amodal 3d object detection. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1742–1749. IEEE, 2019.

[71] Nicolai Wojke, Alex Bewley, and Dietrich Paulus. Simple online and realtime tracking with a deep association metric. *CoRR*, abs/1703.07402, 2017.

[72] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[73] Yanyu Xu, Zhixin Piao, and Shenghua Gao. Encoding crowd interaction with deep neural network for pedestrian trajectory prediction. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[74] K. Yamaguchi, C. A. Berg, E. L. Ortiz, and L. T. Berg. Who are you with and where are you going? *CVPR*, pages 1345–1352, 2011.

[75] Zetong Yang, Yanan Sun, Shu Liu, and Jiaya Jia.

[76] S. Yi, H. Li, and X. Wang. Understanding pedestrian behaviors from stationary crowd groups. In *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 3488–3496, 2015.

[77] Yurong You, Yan Wang, Wei-Lun Chao, Divyansh Garg, Geoff Pleiss, Bharath Hariharan, Mark Campbell, and Kilian Q. Weinberger. Pseudo-lidar++: Accurate depth for 3d object detection in autonomous driving. *CoRR*, abs/1906.06310, 2019.

[78] Fisher Yu, Wenqi Xian, Yingying Chen, Fangchen Liu, Mike Liao, Vashisht Madhavan, and Trevor Darrell. BDD100K: A diverse driving video database with scalable annotation tooling. *CoRR*, abs/1805.04687, 2018.

[79] Jenny Yuen and Antonio Torralba. A data-driven approach for event prediction. *ECCV (2)*, pages 707–720, 2010.

[80] Feihu Zhang, Victor Prisacariu, Ruigang Yang, and Philip HS Torr. Ga-net: Guided aggregation net for end-to-end stereo matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 185–194, 2019.

[81] Pu Zhang, Wanli Ouyang, Pengfei Zhang, Jianru Xue, and Nanning Zheng. SR-LSTM: state refinement for LSTM towards pedestrian trajectory prediction. *CoRR*, abs/1903.02793, 2019.

[82] Tianyang Zhao, Yifei Xu, Mathew Monfort, Wongun Choi, Chris Baker, Yibiao Zhao, Yizhou Wang, and Ying Nian Wu. Multi-agent tensor fusion for contextual trajectory prediction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[83] Bolei Zhou, Xiaogang Wang, and Xiaoou Tang. Understanding collective crowd behaviors:learning a mixture model of dynamic pedestrian-agents. In *IN: PROC. CVPR*, 2012.

[84] Yipin Zhou and Tamara L. Berg. Learning temporal transformations from time-lapse videos. *CoRR*, abs/1608.07724, 2016.