3D Online Multi-Object Tracking for Autonomous Driving

by

Venkateshwaran Balasubramanian

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2019

© Venkateshwaran Balasubramanian 2019

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Abstract

This research work focuses on exploring a novel 3D multi-object tracking architecture: 'FANTrack: 3D Multi-Object Tracking with Feature Association Network' for autonomous driving, based on tracking by detection and online tracking strategies using deep learning architectures for data association. The problem of multi-target tracking aims to assign noisy detections to a-priori unknown and time-varying number of tracked objects across a sequence of frames. A majority of the existing solutions focus on either tediously designing cost functions or formulating the task of data association as a complex optimization problem that can be solved effectively. Instead, we exploit the power of deep learning to formulate the data association problem as inference in a CNN. To this end, we propose to learn a similarity function that combines cues from both image and spatial features of objects.

The proposed approach consists of a similarity network that predicts the similarity scores of the object pairs and builds a local similarity map. Another network formulates the data association problem as inference in a CNN by using the similarity scores and spatial information. The model learns to perform global assignments in 3D purely from data, handles noisy detections and a varying number of targets, and is easy to train.

Experiments on the challenging Kitti dataset show competitive results with the state of the art. The model is finally implemented in ROS and deployed on our autonomous vehicle to show the robustness and online tracking capabilities. The proposed tracker runs alongside the object detector utilizing the resources efficiently.

Acknowledgements

I would like to thank my supervisor, Prof. Krzysztof Czarnecki for providing me the opportunity and continued support and encouragement throughout the program. His guidance and motivation have been pivotal in the success of my work. I would also like to extend my gratitude to all those who helped in making this thesis work possible including my colleagues Erkan Baser, Prarthana Bhattacharyya, Jian Deng and many more at Waterloo Intelligent System Engineering Lab (WISELab).

Dedication

This thesis is dedicated to my parents Balasubramanian K.V. and Uma Krishnan who have always been there to support me, my Guru for showing me the right directions and the Supreme God for his incredible blessings.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Multi-object tracking (MOT) is a critical problem in artificial intelligence and has received great attention due to its widespread use in applications such as autonomous driving, robot navigation, and activity recognition. It is the problem of finding the optimal set of trajectories of objects of interest over a sequence of consecutive frames.

Multi-object tracking is an important task in autonomous driving, where it also leads to many challenges involved in tracking objects belonging to different classes like cars and pedestrians. The objects vary in number across the frames and are subject to occlusion, cluttering, sudden disappearances, changes in appearance due to sensor motion or illumination changes or angle of view or object deformation, abrupt changes in motion, and detection failures. Furthermore, the tracking has to be done in real-time without performance degradation, and the trajectories are used by other critical components in the self-driving system like behaviour planner and local planner that make decisions based on the trajectories.

Traditionally tracking has been performed using conventional algorithms involving filters [50] [20], graphs [89] and network flows [13] [91], to name a few examples. Most of the successful computer vision approaches to MOT have focused on the tracking-by-detection principle [60] [85]. In tracking by detection, the problem is divided into two steps. First, an object detector is used to identify the potential locations of objects in the form of bounding boxes, and then a discrete combinatorial problem is solved to link these noisy detections over time to form trajectories. With the improvement in 3D object detection methods, object tracking in 3D has become an active area of research. In real-time applications like autonomous driving, three-dimensional inference is inevitable to perceive the real-world objects and make decisions accordingly. Finally, online tracking is yet another important

requirement in autonomous driving as the trajectories have to be available immediately when the targets are identified, since we cannot wait longer to make a maneuver that would avoid a collision.

The subproblem of multi-object tracking called *data association* is arguably the most difficult component. Traditional *batch* methods usually formulate MOT as a *global* optimization problem, with the assumption that detections from all future frames are available, and solve it by mapping it to a graph-based min-cost flow algorithm [37, 7]. *Online* Markovian formulations of MOT, on the other hand do not use future frames and often employ greedy or bipartite graph matching methods like the Hungarian algorithm to solve the assignment problem. The success of the final associations is also dependent on the similarity functions used to match the targets and detections. Traditionally cost functions have been handcrafted with representations based on color histograms, bounding box position, and linear motion models [40, 59], but have failed to generalize across tasks and for complex tracking scenarios.

Recently, deep neural network architectures have shown superior performance in many vision-based tasks. Researchers have started leveraging the deep neural network architectures to replace conventional tracking methods. Milan et al. proposed the first end-to-end formulation for MOT, using a recurrent neural network (RNN) to solve the assignment problem for each target independently based on Euclidean cost [56]. However, the use of convolutional neural networks (CNNs) direclty to solve the association problem while also learning the cost function has not yet been investigated.

This thesis proposes FANTrack, an online 3D MOT architecture that models the data association as inference in CNNs. We present a two-step learning approach. The first step learns a similarity function by considering the visual image and 3D bounding boxes as multi-modal input to yield robust similarities. The second step trains a CNN to predict the target assignments based on the computed pair-wise similarities by modeling it as a classification problem. Advantages of this architecture are that it is easy to train, handles the varying number of targets, and it can run in real-time for online tracking. We use selective convolutions to compute the convolution of the features only at the true measurement locations to arrive at the similarity maps, which enables the tracker to run at 20 Hz. Being a learning-based approach it works not only with image and bounding box features, but it can also be extended to other multi-modal input data by extending the network and retraining it. This provides an effective way of sensor fusion for the MOT problem. We use Tensorflow to implement the architectures and perform training and inference. FANTrack is also implemented and deployed in an autonomous vehicle using the Robot Operating System (ROS) [64].

FANTrack is a joint project which I worked with my colleagues Erkan Baser and Prarthana Bhattacharyya. I proposed the Siamese architecture with multi-modal inputs. Erkan Baser helped with designing the cosine similarity and cost function and proposed the idea of the probability of occlusion. Prarthana Bhattacharyya proposed the idea of using CNNs for data association, and we split the implementation among us. I built the Kalman filter and the probability of existence for the tracker and optimized the networks to make them faster by using selective convolutions and other optimization methods. In addition, I implemented the ROS node and tested it in the car.

This thesis is organized as follows: Chapter 2 provides background which covers the concepts used in this research and related work. Chapter 3 describes the actual method in detail describing the Similarity Network and Association network, the dataset and training and experimental results. Chapter 4 gives an overview of the implementation of the ROS node which is deployed and tested in our autonomous vehicle. Finally, Chapter 5 provides the conclusion, the limitations, and future work.

Resources

- FANTrack arxiv paper can be found at https://arxiv.org/abs/1905.02843
- FANTrack source code will be available at https://git.uwaterloo.ca/wise-lab/fantrack

Chapter 2

Background and Related Work

In this chapter, we provide an overview of the various concepts used in this thesis. We begin with the basics of types of learning algorithms and then move into the specifics of deep neural networks and their architectures, the training process and how they are used by the research community to solve the problems in multi-object tracking.

2.1 Machine Learning

Machine learning comes under the broader umbrella of artificial intelligence, where the computer is trained to make inferences on its own by feeding in training dataset. Let's assume that we have the input data $x = \{x_1, x_2, .., x_n\}$ and the expected output is $y = \{y_1, y_2, .., y_n\}$. A supervised machine learning algorithm learns a function f which gives the prediction from the inputs as $y = f(x)$. Machine learning problems that come under supervised learning are further classified into classification and regression problems based on the nature of the predicted output.

2.1.1 Classification

Given the input data x, if the expectation is to classify it into one of the predefined classes $y = \{1, 2, 3, ...n\}$ by learning the mapping $y = f(x)$ then the problem is a classification problem. If the number of classes n is 2, it is known as binary classification and if $n > 2$ then this is known as multi-class classification problem. Typically, in binary classification problems the labels are represented as $y = \{0, 1\}$ or $y = \{-1, 1\}$.

2.1.2 Regression

In case of a regression problem, the machine learning algorithm is made to predict a continuous value in the range $y \in \mathbb{R}$. For example, in a self-driving car, detecting an object in an image and identifying it as a car or a truck is a classification problem, whereas finding out the bounding box coordinates and position estimates of the car is a regression problem.

2.2 Feed Forward Neural Networks

Feed forward neural network is a family of architectures that consist of a collection of interconnected neurons without cycles. These networks model a function f such that $y = f(x, \theta)$ by optimizing a set of parameters θ. They are also used to model composition of functions of the form $f(x) = f^3(f^2(f^1(x)))$ [27]. A feed forward neural network is composed of an input layer, one or more hidden layers, and an output layer. Researchers have come up with various forms of networks depending on the nature of the input data. For example, Convolutional Neural Networks are used for images, and multi-layer perceptrons are used for regression problems involving numerical data. In general, these architectures can be used to solve a non-linear problem, as the relationship between the inputs and the outputs in the network are inherently non-linear [79]. Subsequent subsections describe various components and training techniques used in feed forward neural networks. The structure of a feed forward neural network is shown in Fig. 2.1.

2.2.1 Activation Functions

In a feed forward network, every neuron in the hidden layers is a computational unit which is a non linear function of the inputs fed to it. The non-linear function is given by, $h = f_{NL}(W^T x + b)$, where W is the set of weights w_i for every input value x_i respectively, b is a bias term. These non-linear functions that act upon the linear combination are called activation functions. The structure of a neuron is depicted in Fig. 2.2.

Activation functions control how the subsequent neurons get activated in the network. There are many different activation functions found in the literature, some of which are explained below.

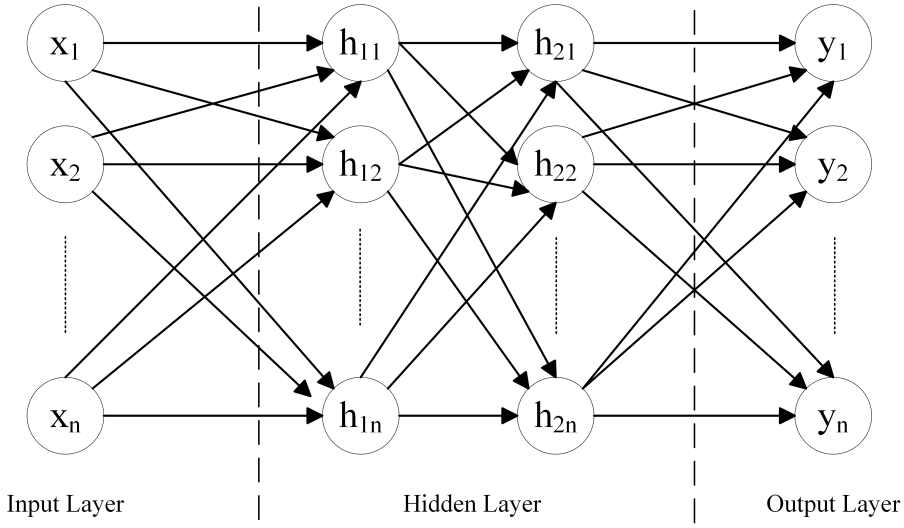


Figure 2.1: Architecture of a Feed Forward Neural Network

Sigmoid function

The sigmoid function gives a smooth non-linearity to the neural network. The formula for a logistic sigmoid function is given by

$$y = \frac{1}{1 + e^{-x}} \tag{2.1}$$

ReLu

Rectified Linear Unit also called as ReLu is another non-linear activation function used in neural networks. It works by clipping the output values that are negative. The output of ReLu activation is given by,

$$y = max(0, x) \tag{2.2}$$

Rectified Linear units are closely related to linear units as they have just two linear pieces, but the function itself is not differentiable at x=0 making it difficult to find the derivatives for gradient-based learning methods [27]. An alternative variant called Softplus

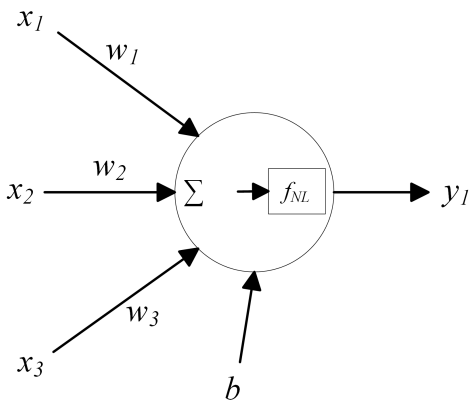


Figure 2.2: Activation function in a hidden neuron

having the formula $y = log(1 + e^x)$ can be used, which makes the function smoother in the regions where ReLu would make sharp cuts.

Leaky ReLu

With ReLu the gradient of the function below 0 is always 0. This causes a problem known as 'Dead ReLu' problem where the gradients in the backpropagation are always zero and the network does not learn anything from these neurons. To solve this problem, Leaky ReLu functions are used. In a Leaky ReLu activation function, the values less than zero are not completely ignored but considered with a fraction. Thus during gradient calculation, a small fraction of these values are preserved thereby solving the dead ReLu problem. The leaky ReLu activation function is given by,

$$y = \begin{cases} x, & \text{if } x > 0 \\ 0.01x, & \text{otherwise} \end{cases} \tag{2.3}$$

2.2.2 Loss Functions

Learning in neural networks takes place by optimizing parameters θ in the model based on the labeled data fed to them. During training, the network is initialized with random weights and then made to predict the output for the training data x by randomly initializing

the set of parameters θ. The difference between the predicted $\hat{y}$ and the ground truth value y is known as loss. The function that models this loss is called as a loss function, which is given by $J(\theta) = g(y - \hat{y})$. Loss functions are chosen based on the problem and the nature of data. For example, L1 and L2 losses are used for regressions problems, whereas binary cross entropy is used for classification. The loss functions used in this thesis are described below.

L2 Loss

L2 loss also known as Mean Squared Error or MSE, in short, minimizes the squared differences between the estimated and the target values. It is also known as Euclidean loss. L2 error tends to amplify the error for outlier data points as the difference is squared [74]. L2 error is given by,

$$J(\theta) = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \tag{2.4}$$

Cosine Distance

The Cosine distance measure is derived from the cosine similarity measure. The cosine distance of two vectors A and B is given by,

$$C_d(A, B) = 1 - \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}} \tag{2.5}$$

Cross entropy loss

Classification models output the probability distribution of the data point belonging to each class and each probability has a range $\hat{y}_j \in [0, 1]$. Cross entropy loss is helpful in modeling such scenarios where the loss function value is continuous and it increases when the predicted probability is away from the actual label. Cross entropy loss is also known as log loss [58] and for a binary classification it can be written as,

$$J(\theta) = -(ylog(\hat{y}) + (1 - y)log(1 - \hat{y})) \tag{2.6}$$

2.2.3 Gradient Descent

Gradient descent is an optimization algorithm used to minimize the cost function by iteratively moving in the direction of the steepest descent. The main difference in using gradient descent algorithm in neural networks when compared to linear or logistic is that the non-linearity of a neural network causes the loss functions to become non-convex [27]. Moreover, the initial weights matter in optimizing the loss function and reaching convergence. In a typical gradient descent algorithm, which is also called as the batch gradient descent, the entire training set has to be iterated over once to finally update the weight values. Sometimes it might not be feasible to load the entire training set into memory. In such cases, variants of the vanilla gradient descent like stochastic gradient descent can be used. In case of the stochastic gradient descent, random samples from the training data set are used to update the weights in each iteration.

2.2.4 Regularization Techniques

The objective of any machine learning algorithm is to model the true underlying function from the given training data. This is done by reducing the generalization error so that the model can perform well on any unseen data point. However, the learning algorithms can be prone to overfitting, in which case, the training loss is less than the validation loss, which in turn affects the generalization capabilities of the model. For example, in curve fitting, a complex polynomial with too many coefficients could fit the data perfectly but may not generalize for data outside the training set. Thus, the complexity of a model determines its generalization capabilities [9]. Regularization is a technique which solves the problem of overfitting by reducing the complexity of the model, leading to better generalization. Conventional machine learning algorithms like linear regression and logistic regression employ a regularization parameter λ in the objective function.

L2 Regularization

In L2 regularization the regularization term is computed as the product of the regularization parameter λ and the squared magnitude of the weights. L2 regularization is also called ridge regression or Weight decay. A typical objective function with L2 regularization is given by

$$J(\theta; X, y) + \lambda \sum_{i=1}^{n} \theta_i^2 \tag{2.7}$$

The regularization parameter λ is a hyper-parameter that dictates the weight given to the regularization term. Too small values reduce the effect of regularization failing to prevent overfitting whereas too high values might lead to underfitting.

Data Augmentation

Deep learning models require huge amounts of training data to arrive at the optimal set of parameters for the model as the number of trainable parameters is huge. At times, it is not possible to get sufficient amounts of real world data for training and as a result, the model could overfit. Data augmentation strategies increase the training set size by producing additional training examples by modifying the data in different dimensions. For example, in case of an image classifier, the data augmentation strategy would be to create augmented samples of a given image by applying image transformations like translation, rotation, and affine transformation as shown in Fig. 2.3. These augmented samples help in modeling the data distribution in real world. The augmented training set helps in reducing overfitting. Hence, data augmentation is one of the most widely used regularization strategies in deep neural networks.

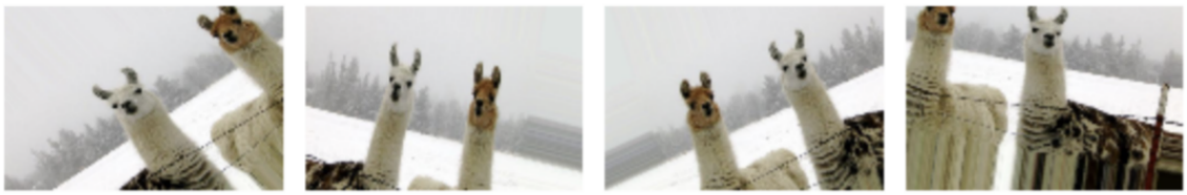


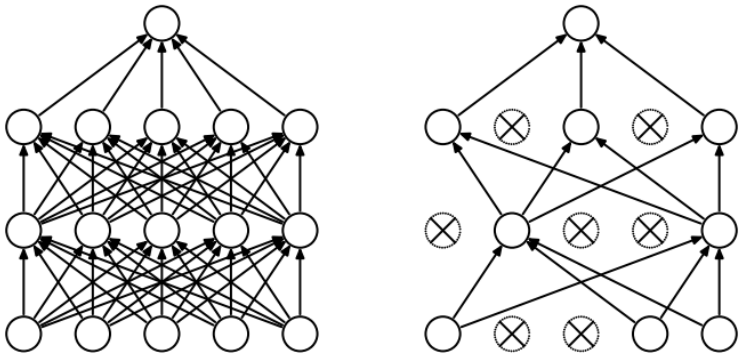
Figure 2.3: Data augmentation with rotation and translation [62]

Early Stopping

During the training of deep neural networks, the loss is expected to decrease with epochs, but sometimes it keeps fluctuating. In this scenario, the loss could actually start increasing after reaching a global minimum. Early stopping is done by saving the network parameters at specific intervals, so that training can be stopped at a specific epoch to get the lowest loss value [9]. This helps in choosing the best snapshot of the parameters.

Dropout

Dropout is a computationally inexpensive but powerful method of regularization typically used in deep neural networks [27]. Often during the training, neurons tend to learn dependent activations from the neighboring neurons, leading to overfitting. To avoid the interdependence of neurons, each iteration randomly drops or discards the activations of some neurons as shown in Fig. 2.4. We denote the dropout rate as $1 - p$, where p is the proportion of neurons to be retained, also known as keep probability. The value for keep probability is one of the hyperparameters for the model. Choosing a very high value will not help in avoiding overfitting and choosing a very low value will cause underfitting. A dropout value of 0.5 is known to be effective [77].



(a) Standard Neural Net (b) After applying dropout.

Figure 2.4: Left: A typical neural network with two hidden layers. Right: Dropout applied by randomly dropping out (crossed) the neurons in the network. [77]

2.3 Convolutional Neural Networks

A convolutional neural network is a type of neural network which handles multi-dimensional data arranged in a grid-like topology, such as images and feature maps. Convolutional neural networks were inspired by neocognitrons [24], which are self-organizing neural network models for pattern recognition. They were first introduced by LeCun et al. [48] to perform handwritten zip code recognition. Later, Krizhevsky et al. [43] developed an end to end image recognition model using CNNs and used them to classify objects in the ImageNet

[17] challenge. Since then, convolutional neural networks have gained momentum in object detection and classification tasks.

2.3.1 The Convolution Operation

The convolution operation gives the product of two signals by reverting one of them and sliding over the other within the given limits. The convolution operation of two real-valued functions $f(t)$ and $g(t)$ is given by,

$$s(t) = (f * g)(t) = \int_{\tau=-\infty}^{\tau=\infty} f(\tau)g(t-\tau)d\tau \tag{2.8}$$

Here we compute the product of f and g on the τ axis after reverting $g(\tau)$ to $g(-\tau)$ and sliding it over $f(\tau)$. For discrete-valued functions, it can be defined as,

$$s(t) = (f * g)(t) = \sum_{\tau=-\infty}^{\tau=\infty} f(\tau)g(t-\tau) \tag{2.9}$$

In case of neural networks, the multidimensional input data, which is in the form of a grid, represents the function f, and a real valued grid of trainable values with predetermined dimensions becomes the sliding function g, which is also known as a kernel. According to Goodfellow et al. [27], convolution leverages three important concepts that are useful in machine learning: sparse interactions, parameter sharing and equivariant representations.

2.3.2 Convolutional layer

A convolutional layer consists of convolutional filters, each with height h, width w, and depth c ($h \times w \times c$). Choosing the dimensions for the filter is a design decision that has to be made according to the input dimensions and the problem. For example, a three channel RGB image will need a three channel filter which has smaller dimensions to generate feature maps with representative power. The filter starts from the first pixel location at the top left of the image and strides across the image producing convolved features. The amount of displacement between the first and the second sample in the convolution operation is called as stride. Fig. 2.5 shows the convolutional filter in action with a filter size of 3x3x1 and stride value 1. The convolutional layer is typically followed by a non linear activation function such as rectified linear units, which produces activation values in the output

feature map. To avoid the reduction of the output height and width dimensions compared to the input image, the image is padded with zeros at the borders with a specific amount. The final output size (height or width) of the resulting feature map is $(W - F + 2P)/S + 1$, where W is the size of the input image, F is the filter size, S is the amount of stride, and P is the amount of padding used [38].

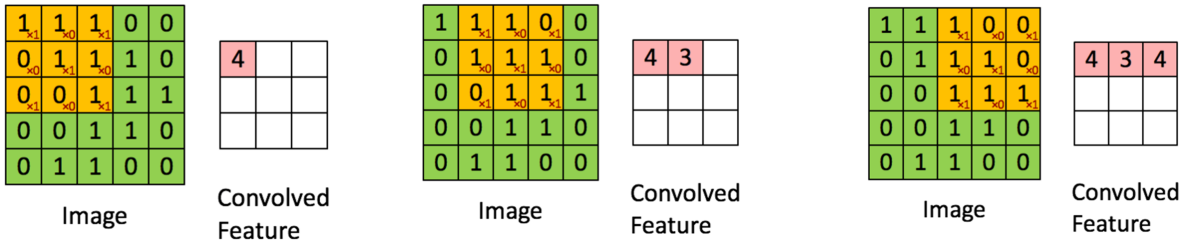


Figure 2.5: A convolutional layer with filter size 3x3x1 and stride 1 (red subscripts) operating on a 5x5 single channel image (shown in green). [71]

2.3.3 Pooling

The output of the convolutional layer is often connected to a pooling layer. The pooling layer downsamples the output of the previous layer by applying a summary statistic. Typical pooling operations include maximum pooling, average, and L2 norm. Pooling is performed for multiple reasons. It reduces the dimensionality of the data to reduce overfitting. It is also helpful in making the network invariant to small transformations in the input. For example, if we have trained the network to detect a car in the image, during inference the exact location of the car can vary anywhere in the image and the network should not be overfit to detect at certain locations. A typical max pooling operation is shown in Fig. 2.6.

Global Average Pooling

Global Average Pooling layer was first proposed by Lin et al. in [52]. In their work they proposed a novel architecture called "Network In Network" replacing the CNNs with micro networks to enhance model discriminability within the receptive field. Instead of using a fully connected layer in the final classification layer they used global average pooling over the feature maps and proved that it reduces overfitting. The idea behind global average

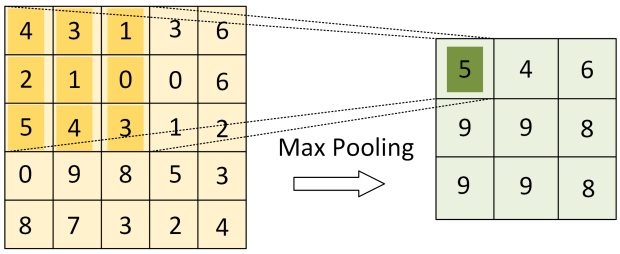


Figure 2.6: Max pooling operation with size 3x3x1 on a 5x5 input image

pooling is to obtain one feature map for every category in the classification task by taking the spatial average of the corresponding channel of the feature map and taking the softmax of the resulting layer to obtain the final classes. This brings the convolutional structure to the classification task and removes the need to train additional neurons in fully connected layers. This also helps in making the network robust to spatial translations in the input. An example of global average pooling over a 6x6x3 feature map is shown in Fig. 2.7.

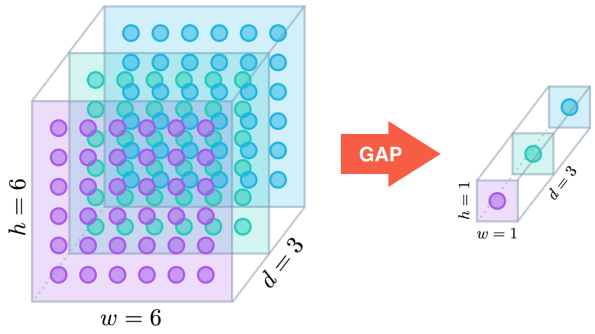


Figure 2.7: Global average pooling over a 6x6x3 feature map. [15]

2.3.4 CNN as a feature extractor

A convolutional neural network is a hierarchical feature extractor. For example, in an image classification network, every convolutional layer extracts a certain level of hierarchy on the goal to fully identify the object in the image. For an object detector to identify a car, it needs multiple features with respect to the wheels, the body of the car and the placement of

these at the corresponding places. These information are learned in a hierarchical manner in the different layers in the network. A typical image classification model like VGG16 [75] has convolutional layers that can identify important features in an image in a hierarchical manner and this can be utilized for a different task like comparing the visual similarity of two objects.

2.3.5 Dilated Convolution

A dilated convolutional layer is one in which the convolutional filter is applied with a dilation factor. In a regular convolution operation, the convolutional filter is applied on the input image such that the consecutive pixels on the image are considered in the computation of the output value of the convoltion. In case of dilated convolution, the filter is applied such that it considers the pixels with an interval in between, thereby operating on a bigger area on the input image. The extent of this interval is parameterized as the dilation factor. Dilated convolutions support exponential expansion of receptive field on the input image. Dilated convolutions were first proposed in the CNN architecture by Yu et al. [88]. The authors use dilated convolutions to aggregate multiscale contextual information without loss of resolution or coverage and use it for the semantic segmentation task. The dilated convolutional filters are applied with increasing dilation factor which helps in exponentially increasing the receptive field. Fig. 2.8 depicts the difference between a normal convolution operation and a dilation convolution with dilation factor d=2.

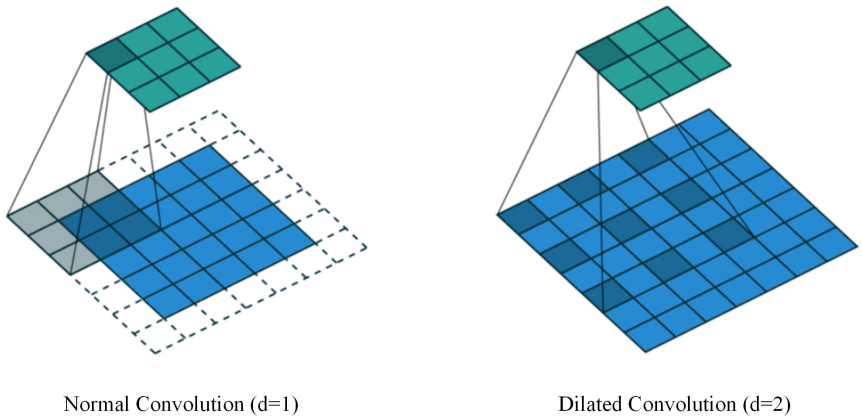


Figure 2.8: Difference between the standard convolution operation and dilated convolution operation. [82]

2.4 Siamese Network

A Siamese network has two identical branches which share the same weights. Two inputs are fed separetely to the branches and the output is an embedding. The embedding layer is used in the loss function and is typically used to find the similarity between the two inputs. The generic architecture of a Siamese network is shown in Fig. 2.9. Siamese networks were first introduced by Bromley et al. [12] in 1994. The authors used the Siamese architecture to identify the similarity between two signature patterns to verify the originality of the second signature. A set of handcrafted features from the signatures were chosen as input and time delay networks were used as the core network architecture in the Siamese arrangement. The extracted features from the networks were used to compute the cosine of the angle between the two vectors which represents the similarity value. The two key properties [31] of Siamese networks that enable them to be suitable for similarity computation are:

1. Weight sharing ensures two similar input samples do not map to different locations in the embedding space. This aids in identifying the similar samples.
2. The network is symmetric so that the order of the inputs does not matter.

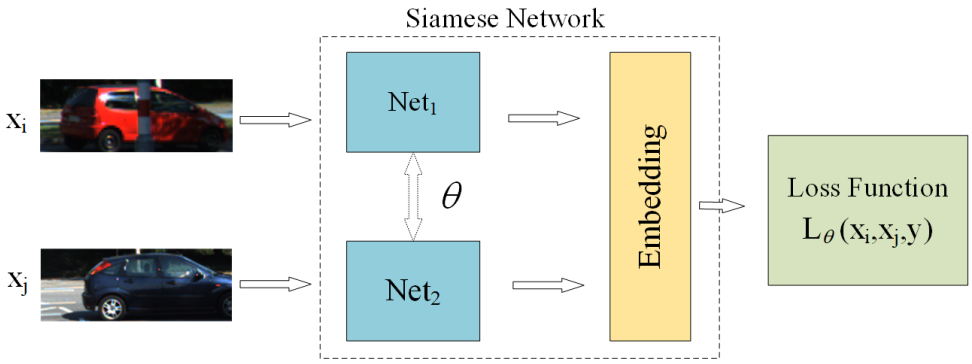


Figure 2.9: Generic architecture of a siamese network.

Siamese networks have recently been used for computer vision tasks with the advent of convolutional neural networks. Chopra et al. [14] use convolutional neural networks in a Siamese architecture for a similarity metric for face verification. They have used contrastive loss function which minimizes the similarity metric for the same person and increases it for

different faces. Koch et al. [42] use Siamese networks for one shot image recognition, which is able to increase the predictive power of the network for new classes from new data distributions. Varior et al. [83] define a gating function in the Siamese architecture to propagate the mid-level features learned by the CNN, which helps in improving the discriminative power of the network. They apply this idea to the task of human re-identificaiton in videos. Hoffer and Ailon [34] propose a triplet network consisting of three networks which accept three inputs. They address the problem of calibration in Siamese networks by computing the distance metric based on the intermediate representation of the two inputs and comparing with the third. Leal-Taixe et al. [47] use CNNs in Siamese fashion to come up with a matching cost for data association using optical flow and spatio-temporal structures. Tao et al. [80] propose an object tracker which uses Siamese network for modeling the matching function which is invariant to distortions using instance search paradigm. The first stream of the network is called as query stream, which represents the target object derived from a random video frame. The second stream is known as the search stream, which is another bounding box from a different frame later in the video. Depending on the overlap with the ground truth, positive and negative examples are defined. The authors have experimented with both a VGGNet and AlexNet like architecture for the core network.

2.4.1 Loss Function

Unlike traditional neural network architectures, the loss function used in Siamese networks has to be designed differently. Usually, loss functions in neural networks are minimized using gradient descent techniques so that the network learns from the data. Let's consider a Siamese network which computes the similarity between two image patches x_1 and x_2. Let y be the binary label assigned to the examples: $y = 0$ if the inputs are dissimilar and $y = 1$ when the inputs are similar.

We design two different loss functions L_p and L_n for positive and negative example pairs respectively and minimize the loss L_p and maximize the loss L_n so that the network will be able to discriminate between the positive and negative scenarios. The combined loss is given by

$$L(\theta) = \sum_{(x_{p1}, x_{p2})} L_p(x_{p1}, x_{p2}) - \sum_{(x_{n1}, x_{n2})} L_n(x_{n1}, x_{n2}) \tag{2.10}$$

If we consider the Euclidean distance as D^i for the ith example then according to [30] we can write the loss function as,

$$L((x_1, x_2)^i, y) = (1 - y)L_p(D^i) + yL_n(D^i) \tag{2.11}$$

L_p and L_n must be designed such that minimizing L with respect to θ would result in low values of D^i for similar examples and high values of D^i for dissimilar examples. This loss function is called contrastive loss function as it is able to differentiate between the similar and dissimilar examples.

2.4.2 Siamese Topologies

Leal-Taixe et al. [47] discuss various topologies for processing the inputs using Siamese CNNs as shown in Fig. 2.10.

Cost Function

In the Cost Function method, the two input images are separately processed by two sets of convolutional layers which share the same weights. Interaction between the input pairs do not happen till the cost function where the encoding in the latent space are used to compute the loss.

In-network

In this method the two input image patches are separately processed by few layers and then interaction between the pairs begins to happen in the later layers before the cost function is applied.

Input stacking

In input stacking the two input patches are stacked together and form a unified input to the CNN.

According to Leal-Taixe et al. [47], the input stacking method is known to produce good results but the cost function approach is much faster.

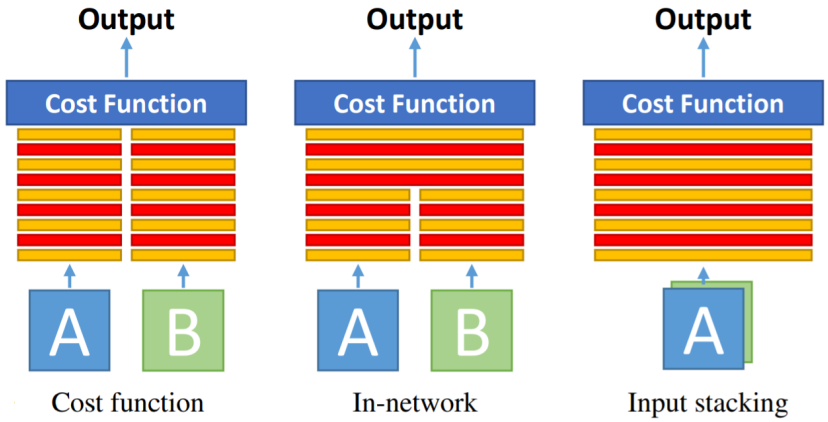


Figure 2.10: Siamese CNN Topologies [47].

2.5 Kalman Filters

In control theory, state estimation is the problem of identifying the optimal state of the system based on the measurements from the sensors by filtering the noise induced in the readings. State estimation has two main problems for a discrete system with $\hat{x}_k$ as the state at time k .

1. $\hat{x}_{k|0..k}$: To find the current state, given a set of measurements upto time k.
2. $\hat{x}_{k+1|k}$: To find the predicted state at time $k+1$ given the measurement at the current time instant k.

A Kalman filter is an optimal linear estimator which is able to provide solutions for both problems using a recursive algorithm. The process of removing the noise in the data to arrive at more accurate readings is called filtering. Kalman filter was introduced and named after Rudolf E. Kalman in 1960. Multiple extensions of the filter have been defined to handle non-linear systems.

The Kalman filter gives us a probabilistic estimate of the actual state of the system, represented as Gaussian mean $\hat{x}_k$ with an error covariance of $P_{k|k}$.

Now let z_k be sensor measurement at time k. We model the uncertainty in the measurement as a Gaussian distributed noise with zero mean and covariance R_k. Thus, this

noise is $v_k \sim \mathcal{N}(0, R_k)$. The actual state of the system in the next instance is modeled as a random process as there is some uncertainty about the next measurement. The randomness is modeled as process noise, which is Gaussian distributed with zero mean and a covariance Q_k, that is the process noise is $w_k \sim \mathcal{N}(0, Q_k)$. The equation to derive the state of the system at time k from time $k-1$ is as follows

$$x_k = F_k x_{k-1} + B_k u_k + w_k \tag{2.12}$$

where F_k is the transition matrix which models the state transition from $k-1$ to k, B_k is the input matrix and u_k is the input vector. For instance, a Kalman filter which is designed to estimate the position and velocity of a moving object will have a transition matrix modeled using the Kinematic equations for motion.

The measurement z_k is modeled as follows

$$z_k = H x_k + v_k \tag{2.13}$$

using the observation matrix H, which relates the measurement vector to the state vector, and v_k models the measurement noise as stated earlier.

The Kalman filter has two stages: prediction and correction, which are repeated iteratively.

Prediction Stage

During the prediction stage, the state of the system is predicted using the state transition matrix F_k. The error covariance P_K of the state is also updated using the state transition matrix and the process noise covariance Q_k. This implies that we are trying to get a predicted value for the state using the motion model we defined and also predict the error based on the error in the previous state and our assumption of the random process noise we applied. The equations for prediction are given as,

$$\hat{x}_{k|k-1} = F_k \hat{x}_{k-1|k-1} + B_k u_k \tag{2.14}$$

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{2.15}$$

Correction Stage

During the correction stage, we use the measurement from the sensors z_k and correct the predicted state to arrive at the estimated value of the state. We calculate a parameter called Kalman gain K_k, which is calculated from the predicted error covariance $P_{k|k-1}$, the measurement model H_k and the measurement noise covariance R_k as,

$$K_k = P_{k|k-1}H_k^T(H_kP_{k|k-1}H_k^T + R_k)^{-1} \tag{2.16}$$

Later we also update the state estimate and the state error covariance $P_{k|k}$ as

$$\hat{x}_{k|k} = \hat{x}_{k|k-1} + K_k y_k \tag{2.17}$$

$$P_{k|k} = (I - K_k H_k)P_{k|k-1} \tag{2.18}$$

where y_k is the residual, which is the difference between the actual measurement and the projected measurement from the predicted state:

$$y_k = z_k - H_k\hat{x}_{k|k-1} \tag{2.19}$$

Fig. 2.11 summarizes the Kalman filter stages by plotting the probability density functions of the previous state, the predicted state, the state projected from the measurement, and the resulting current state [39] by representing the actual and the predicted state estimates and the measurements as a probability density function.

2.6 Multi-object tracking

Multi-object tracking is one of the most vital tasks in autonomous driving. The task is to keep track of the objects in the neighborhood of the self driving car and produce the trajectories for them, which allows subsequent modules to make safe planning decisions. In visual tracking, images from the camera are used to detect and track the objects. With the advent of lidar and other types of sensors, sensor fusion techniques [66] [57] are becoming important to fuse the data from multiple sources. Visual single and multi-object tracking have remained difficult due to the various challenges such as occlusion, detector noise, clutter, target reidentification, deformation, and lighting changes.

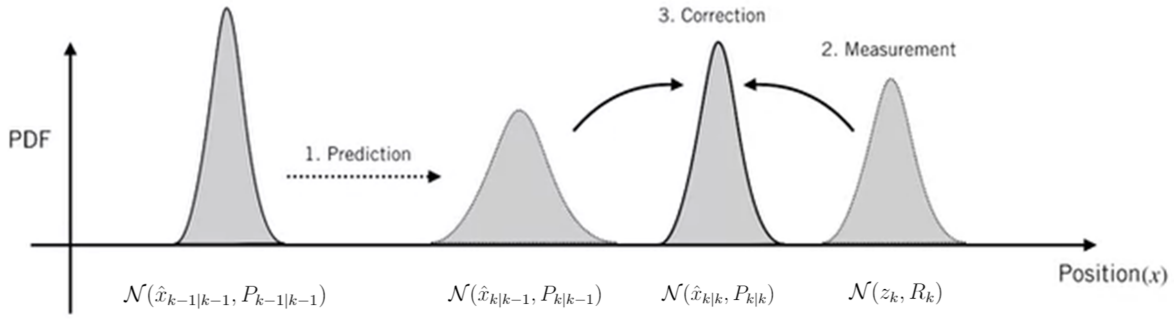


Figure 2.11: Visualization of the Probability density function of a state estimate using Kalman Filter [39].

Luo et al. [54] provide a comprehensive survey of multi-object tracking, which they subdivide based on initialization strategy, processing and the output of tracking. Based on initialization strategy, trackers can be classified as detection-based tracking and detection-free tracking. Based on the processing mode, trackers are further classified as online and offline. Finally, based on their outputs, trackers are further classified as deterministic and probabilistic.

Detection-based tracking

Detection-based trackers depend on an object detector for the bounding boxes to track the objects. An object detector is used to identify the potential locations of the objects, their dimensions and classes and then tracking is modeled as a discrete combinatorial problem to link the noisy detections to form trajectories or tracks. One of the main drawbacks of this strategy is that the accuracy of the tracking is limited by the accuracy of the object detector used. Work by Xiang et al. [86] is an example of a tracking-by-detection strategy, which models the tracker states as a Markov Decision Process.

Detection-free tracking

In detection-free tracking, an object detector is not explicitly used to get the bounding boxes for tracking. Instead they either require manual initialization of the targets [90] or have the detection mechanism ingrained in the tracking pipeline [1], also known as joint

detection and tracking. Joint detection and tracking approaches are further classified into those that only model the object's appearance [68] and those that model both the object's appearance and its background [28].

Learning-based joint detection and tracking

Learning-based joint detection and tracking uses a single network to do both object detection and tracking. End-to-end approaches have the advantage of having a single loss function that simultaneously optimizes the detection and tracking, which has shown to produce better results [23], when compared to piece-wise approaches. End-to-end approaches also have the advantage of reusing the appearance features and object priors [21]. However, end-to-end approaches are prone to error propagation [92].

Online Tracking

Online Tracking uses only the information available up to the current frame to assign the tracks to the objects. In other words, the tracker performs a real time tracking without any post processing tasks. Online trackers are more challenging to build as they do not have the advantage of looking into the future observations. Various online tracking approaches exist in the literature. Sharma et al. [73] use geometry and object shape costs to build an online tracking algorithm. Gunduz et al. [29] propose an online tracker based on min-cost linear cost assignment by affinity matching. Tian et al. [81] design an online tracking algorithm using tracking-by-detection paradigm. The detections are combined to form tracklets and the association of tracklets to tracks is performed using the structural information on a motion pattern between the objects. Scheidegger et al. [72] train a deep neural network to obtain 3D coordinates of tracked objects from camera images and build a Poisson multi-Bernoulli mixture tracking filter for the tracking algorithm. Xiao et al. [87] propose a joint probabilistic relation graph approach to track objects online from aerial videos.

Offline Tracking

Offline tracking algorithms look at the entire batch of frames in the sequence and analyze them jointly to estimate the output. The drawback with offline tracking is that they cannot be used in a real-time system or in systems where delays are not acceptable. On the other hand, offline tracking algorithms are comparatively easier and more accurate in generating the trajectories as they access both the past and future observations, to infer the

current frame. Wang and Fowlkes [84] propose a framework for offline tracking by learning the parameters for min-cost flow by using structured predictions with a tracking-specific loss function. Brendel et al. [11] formulate the data association problem as finding the maximum-weight independent set of a graph. Song et al. [76] solve the problem of maintainability of longer tracks by analyzing the statistical properties of the shorter tracklets. Qin and Shelton [63] formulate the pedestrian tracking problem as a clustering problem and come up with a nonlinear global optimization problem to maximize the consistency of visual and social grouping behavior for trajectories.

2.6.1 Data Association

The data association problem forms the core of the multi-object tracking problem. At time instant τ, let us consider a set of objects that have been observed n the past, that is, from the first frame until τ. The set of trajectories corresponding to these objects are called targets or tracks (T). The set of objects observed at time τ are called measurements (M).Data association finds assignments of the form (t_i, m_j), where $t_i \in T$ and $m_j \in M$ are the i_{th} target and j_{th} measurement, respectively, and not two tracks are assigned the same measurement. This is done by minimizing a cost function or maximizing the similarity metric between the corresponding target and the measurement. Data association gives us pairs that have the highest similarity scores, which helps in the encompassing problem of tracking.

Probabilistic Data Association

Probabilistic data association is a Bayesian strategy that associates the target being tracked with potential measurements by assigning probabilities which account for the measurement uncertainty. Shalon et al. [4] proposed a probabilistic data association filter which functions as an optimal state estimator in the presence of uncertainty in the data association. The filter functions similar to the Kalman filter with a few differences. PDAF has a selection procedure for the measurements, unlike a simple Kalman filter which typically uses a single measurement. It chooses the best measurement for the filter update based on the probability, used as a weighting factor. The PDAF algorithm also assumes that there will be a single measurement corresponding to the target that has the highest probability of association and all other candidates of the measurements will be treated as noise. Let us consider the state of the target $x_{k|k}$ and the measurement z_k from the Kalman filter formulations (2.12 and 2.13). From equation 2.17 we get the state $x_{k|k}$ of the target. The state estimate [49] of the PDAF is given by 2.20 with the assumption 2.21.

$$\hat{x}_{k|k} = E^*[x(k)|Z^k] \tag{2.20}$$

The pdf of the target state is approximated as a single Gaussian with mean $\hat{x}(k|k-1)$ and covariance $P(k|k-1)$ as follows,

$$p[x(k)|Z^{k-1}] = \mathcal{N}[x(k); \hat{x}(k|k-1), P(k|k-1)] \tag{2.21}$$

Here $E^*[x(k)|Z^k]$ gives the optimal estimate in which Z^k denotes the set of all measurements observed through time k. Now, we look at the formulations accounting for the difference the PDAF has compared to the Kalman Filter. The combined innovation, which is also the measurement residual $y(k)$ in the Kalman filter is given by 2.22

$$y(k) = \sum_{i=1}^{m(k)} \beta_i(k) y_i(k) \tag{2.22}$$

$\beta_i(k)$ is the association probability corresponding to the ith candidate of the mesurement and the target at time k, which is given by

$$\beta_i(k) = \begin{cases} \frac{\mathcal{L}_i(k)}{1-P_D P_G + \sum_{j=1}^{m(k)} \mathcal{L}_j(k)} & i = 1, .., m(k) \\ \frac{1-P_D P_G}{1-P_D P_G + \sum_{j=1}^{m(k)} \mathcal{L}_j(k)} & i = 0 \end{cases}$$

Here the first case is a regular case where the association exists, and the second case where $i = 0$ is when there is no assignment possible. P_D is the probability of detection and P_G is the gate probability, which is the probability that there is a measurement whithin a predefined threshold of space around the target, also called as gate. The term $\mathcal{L}_i(k)$ is the likelihood ratio of the measurement originating from the target rather than from clutter [4]. It is modelled using a Poisson process [3].

The modified form of the error covariance of the state estimate in the filter incorporating the association probability $\beta_i(k)$ is given by

$$P_{k|k} = \beta_0(k) P_{k|k-1} + (1 - \beta_0(k)) P^c_{k|k} + P^u_k \tag{2.23}$$

The term $P_{k|k-1}$ corresponds to the case when there is no measurement possible for the given target, the term $P^c_{k|k}$ corresponds to the covariance when there is one possible measurement association, and the last term P^u_k handles the uncertainty in choosing the right measurement when multiple measurements are available.

Joint Probabilistic Data Association

Probabilistic Data Association deals with single object tracking. Joint Probabilistic Data Association (JPDA) extends it to multiple targets by jointly computing their association probabilities. JPDA evaluates conditional probabilities of the join events χ [22]:

$$\chi = \cap_{j=1}^{m} \chi_{jt_j} \tag{2.24}$$

where χ_{jt_j} is the event that the measurement j originated from the target t_j, and t_j is the index of the target associated with the measurement j. Only the joint events in which no more than one measurement could be associated with a given target are considered feasible. The state estimation of the individual targets is done independently by assuming the measurements as mutually independent or jointly either by computing cross covariances which represent the cross correlation between the estimation errors of the targets [4]. Many variants of the JPDA algorithm exist in the literature. Svensson et al. [78] propose a modified JPDA filter using Random finite sets. Blom et al. [10] propose a JPDA filter algorithm that modifies the way the association probabilities are computed between the measurements and targets. Drummond [19] uses the Global Nearest Neighbour approach in JPDA.

Multiple Hypothesis Tracking

The PDA and JPDA algorithms focus on developing target-oriented hypotheses. In contrast multiple hypothesis tracking algorithms generate hypotheses based on the measurements. Each measurement is associated with every target to form a cluster of tracks which form the set of possibilities. Finally, only the track hypotheses that are logically possible (where given measurement does not appear in another hypothesis) are chosen and are collectively considered as a global hypothesis (which is a set of track hypotheses that are not in conflict). The highest scoring hypothesis is used to finalize the tracks. In this approach, the actual data association and the track assignments are delayed until the final hypothesis is selected. The original multiple hypothesis tracking algorithm [65] was applied to radar tracking, which took more processing time and it was considered inefficient for visual tracking. But Kim et al. [40] revisit the approach by incorporating appearance models and show that it has comparable performance to that of other state-of-the-art methods.

Hungarian algorithm

The General Assignment Problem is defined as follows. Given an $n \times n$ matrix $R = (r_{ij})$ of positive integers, the goal is to find the permutation $j_1, ..., j_n$ of integers $1, .., n$ that maximizes the sum $r_{1j_1} + ... + r_{nj_n}$. The dual of this problem is to find the non-negative integers $u_1, ..., u_n$ and $v_1, ..., v_n$ such that

$$u_i + v_j \geq r_{ij} \quad i, j = 1, .., n \tag{2.25}$$

which minimizes the sum $u_1 + ...u_n + v_1 + ... + v_n$.

The Hungarian Assignment algorithm, also known as Kuhn-Munkres algorithm, is a combinatorial optimization algorithm that solves the assignment problem in polynomial time [45]. Dell'Amico et al. [16] provide a summary of the Hungarian algorithm as follows.

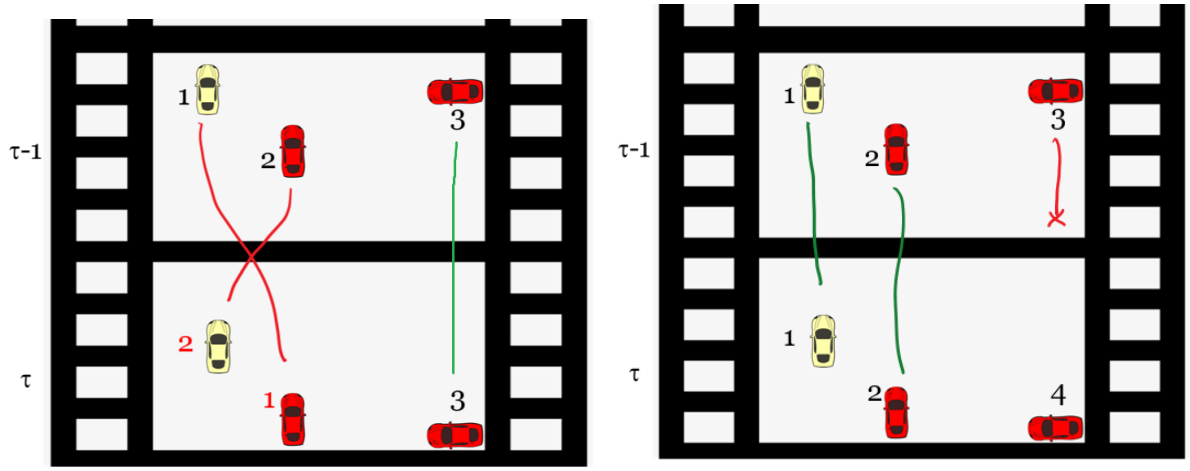
1. Find a dual feasible solution (u, v) using row and column reduction.
2. Given the solution (u, v) solve the restricted primal problem.
3. If the primal solution is feasible then stop. Otherwise, find a new dual solution by setting $\hat{u_i} = u_i + \delta \ \forall i \in [1, n]$ and $\hat{v_i} = v_i - \delta \ \forall j \in [1, n]$, where δ is the minimum reduced cost $\hat{c}_{ij} = c_{ij} - u_i - v_j$. Set $u = \hat{u}$ and $v = \hat{v}$ and go to step 1.

Multi-Object Tracking Metrics

Overall performance of a multi-object tracking algorithm is determined with the help of the CLEAR MOT metrics [8]. The performance of the tracking is measured by two important metrics: Multi-Object Tracking Precision (MOTP) and Multi-Object Tracking Accuracy (MOTA). MOTP is given by

$$\text{MOTP} = \frac{\sum_{i,\tau} d_t^i}{\sum_\tau c_\tau} \tag{2.26}$$

and it measures the total position error for the associated hypotheses over all frames averaged by the total number of matches made. Here d_τ^i is the distance between the i^{th} associated hypothesis and the ground truth detection, and c_τ denotes the total number of associations made at time τ. It represents how precise are the position estimations for the matched target-measurement pairs over all the frames, averaged by the total number of associations made. Multi-Object Tracking Accuracy (MOTA) is given by



(a) ID Switching scenario (b) Fragmentation scenario

Figure 2.12: Illustration of track switches and fragmentation

$$\text{MOTA} = 1 - \frac{\sum_\tau (fp_\tau + fn_\tau + \text{IDS})}{\sum_\tau g_\tau} \tag{2.27}$$

where fp_τ denotes the number of false positives, fp_τ denotes the number of false negatives, and IDS denotes the number of ID switches. An ID switch is a case when two tracks interchange their IDs when observed at two time instants $\tau - 1$ and τ as shown in Fig. 2.12a.

Mostly Tracked (MT), Mostly Lost (ML), and Fragmentation (FRAG) are three additional metrics [51]. Mostly Tracked is given by the proportion of tracks tracked for more than 80% of their lifetime. Mostly Lost is given by the proportion of tracks tracked for less than 20% of their lifetime. Fragmentation denotes the total number of times a ground truth trajectory is interrupted in the tracking result (Fig. 2.12b).

Chapter 3

Method

In this chapter, we describe our proposed tracker: "FANTrack" [5] comprising the Simnet and AssocNet architectures for data association, and Kalman filter and track update strategies. Our problem setup assumes that at any time instant τ we have N targets and M measurements. An overview of the architecture is shown in Fig. 3.1. We use AVOD [44] as our 3D object detector, but in principle, any other 3D object detector could be used.

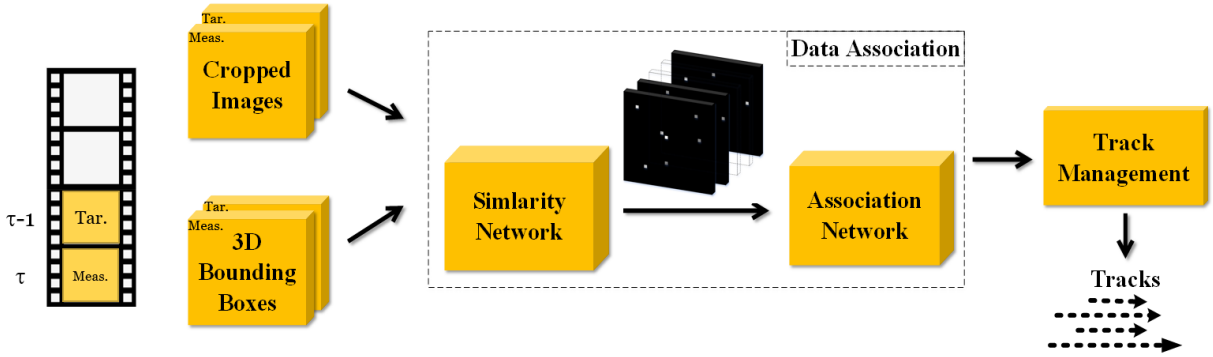


Figure 3.1: Overview of the proposed approach. The targets are from frame $\tau - 1$ and Measurements are in frame τ.

3.1 Similarity Network

The similarity network, also called as SimNet is the first network in our combined architecture for data association (Fig. 3.2). The main role of the network is to compute a similarity metric between every target and measurement pair and to structure the results as localized maps, which are then used by another network for data association. The SimNet uses three instances of a Siamese Network architecture, one for each of the two types of features (i.e., bounding boxes and appearance), and a third one for weighting the relative importance of each feature. The input to the SimNet consists of bounding box parameters and appearance features of the targets and measurements. We follow the approach of combining each of the two Siamese branches in the cost function [47] [18], by separately combining the target and measurement specific inputs. Though this approach leads to slightly reduced accuracy [47], it is faster than the approach of input stacking. The *output* from *SimNet* is a set of maps called Local Similarity maps with dimensions 21x21 and N_{max} channels, one for each target from the $t-1$ frame. Each map is centered around the corresponding target and records the similarity score of the target with every measurement within the $10\,\text{m} \times 10\,\text{m}$ bird's eye view region around the target. The resolution of the maps is $0.5\,\text{m}$.

The SimNet has two main branches: a *bounding box branch* and an *appearance branch*. Each of the two branches, being a Siamese network, applies the same set of weights to each of its input, i.e., target or measurement representation, separately and outputs the corresponding normalized feature vector for each of these inputs. Their respective contribution towards the final similarity score computation is weighted using the *importance branch*. Finally, cosine-similarities of each target-measurement unit vector pairs are computed and the scalars are mapped to their corresponding positions on the above-mentioned set of local maps. We describe the individual branches and the similarity and map generation processes in detail in the subsequent subsections.

3.1.1 Bounding Box Branch

The bounding box branch outputs normalized vectors, each representing a target or measurement bounding box. The cosine similarity between any one of the target unit vectors and any one of the measurement unit vectors is obtained as the dot product of the two vectors. We train a Siamese network with input pairs of target and detection 3D bounding boxes for this purpose. The 3D bounding boxes are defined by their centroids (x, y, z), dimensions (l, w, h), and rotation around the z-axis (θ_z) in the ego-car's IMU/GPS coordinates. To prevent learning variations induced due to ego-motion, detection centroids

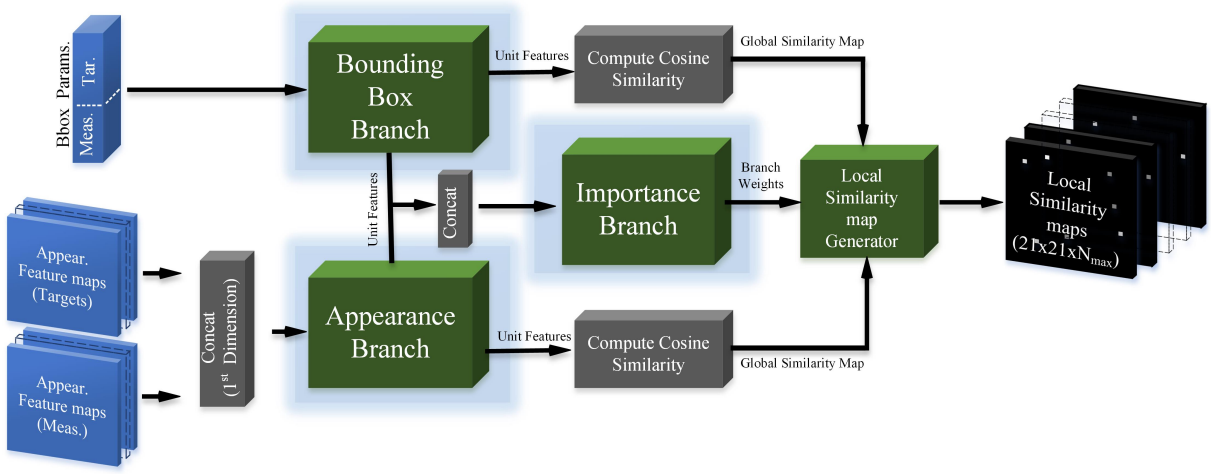


Figure 3.2: Architecture of SimNet. The branches highlighted in blue have trainable parameters. Each one of them is a Siamese network, applying the same set of weights to each target or measurement input.

are converted to coordinates at a common time-step using GPS data. The bounding box parameters are obtained from the object detector (AVOD) and are represented as follows,

$$[x, y, z, l, w, h, ry] \tag{3.1}$$

The input to this branch is an $(N+M)\times 1\times 7$ tensor where the third dimension consists of the 7 bounding box parameters defined above. The inputs are fed to a convolutional layer with 256 1×1 filters to capture complex interactions across the 7 channels, followed by two convolutional layers with 512 1×1 filters, by processing the parameters of each target and detection independently [52]. We apply L2 normalization on the output features and henceforth refer to the result as *unit* features. The unit features have dimensions $(N + M) \times 512$. These unit features are sliced on the first dimension according to the number of targets T to get target and measurement bounding box features respectively. The slicing of the unit features into target and measurement specific features is shown in Fig. 3.4. These sliced unit features along with those obtained from the appearance branch are used to compute the cosine similarities, which will be discussed in the subsequent sections. We use batch normalization and leaky-ReLU activation across all the layers. A detailed architecture of the bounding box branch is shown in Fig. 3.3

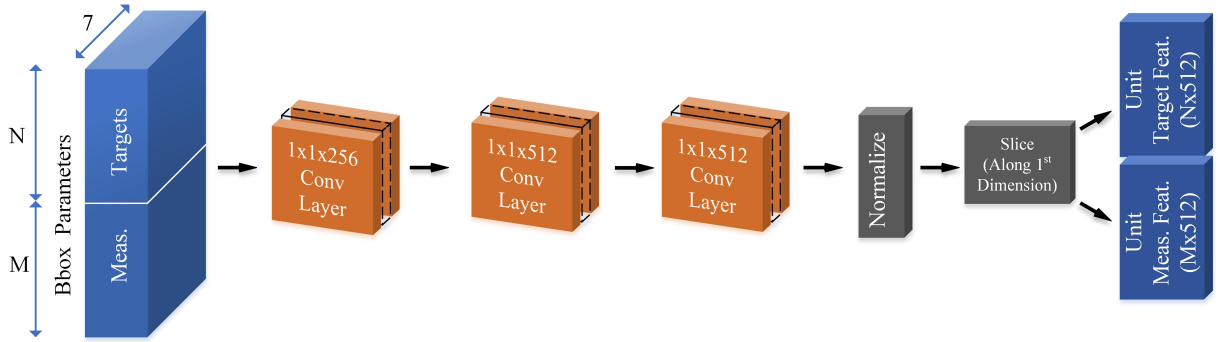


Figure 3.3: Detailed architecture of the bounding box branch. The inputs are bounding box parameters of N targets and M measurements. In training, N and M are 128 (batch size) respectively.

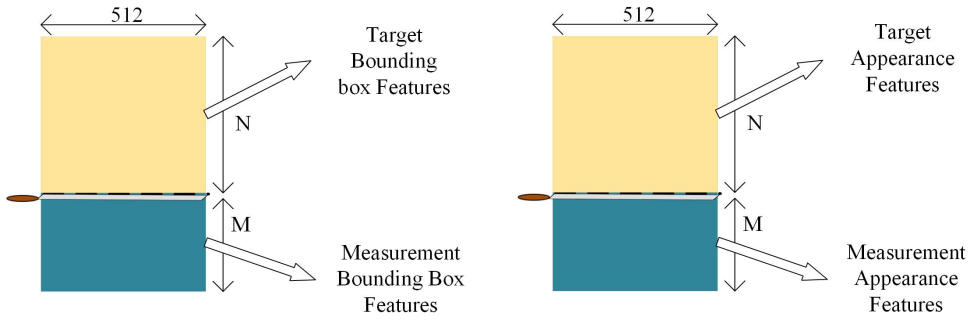


Figure 3.4: Slicing unit features to obtain target and measurement specific features.

3.1.2 Appearance Branch

The appearance branch is used to analyze the 2D visual cues in the targets and measurements for the computation of the similarity scores. We train another stacked Siamese network for this purpose. The input to this branch are the appearance features which are the feature maps which are the output of the convolutional layers from VGG16 pre-trained on ImageNet dataset. The dimensions of the input appearance feature maps are (7x7x640). The architecture of the branch is shown in Fig. 3.5. First, we apply 256 3×3 convolutions to obtain promising features for similarity learning by preserving the spatial size of the input. Before flattening the feature maps for the fully-connected layers with 512 neurons, the Global Average Pooling (GAP) [52] layer extracts one abstract feature from

each feature map. Similar to the bounding box branch, L2 normalization yields a vector of dimension $(N+M)\times 512$. As in the case of the bounding box branch, the $(N+M)\times 512$ features are sliced along the *first* dimension (as shown in Fig. 3.4) to obtain appearance features of detections and targets to compute the appearance cosine similarities.

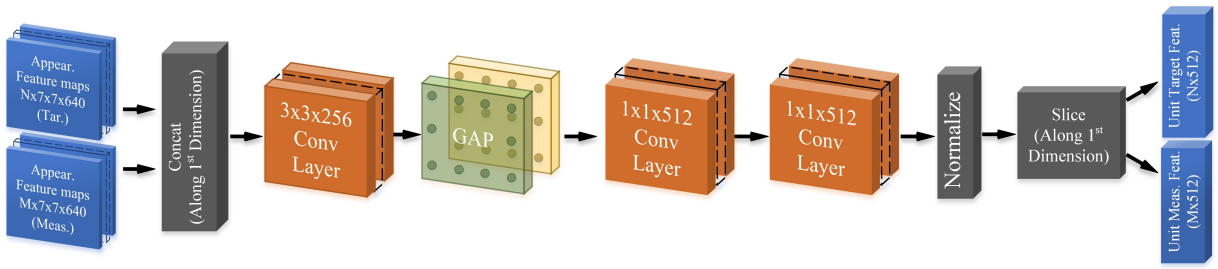


Figure 3.5: Detailed architecture of the appearance branch.

3.1.3 Importance Branch

The aim of the importance branch [32] is to determine the *relative importance* of the bounding box and appearance features in the computation of the final cosine similarity score. The inputs to this branch are the unit bounding box and unit appearance features of both targets and measurements (Fig. 3.6). First, the vector representation of objects obtained from the appearance and bounding box branches are concatenated to form a single vector (dimension 1024). This, in turn, is connected to a fully-connected layer having two neurons. Finally, the softmax layer computes two scalars representing the importance weights as probabilities of the two branches. The weights are sliced along the first and second dimensions to obtain the individual target t and measurement m, specific weights for both the branches. Using these individual weights of the two branches for a given target and measurement their branch-computed similarity scores are aggregated as follows:

$$\omega_\beta = \frac{\omega_\beta^{(t)} \times \omega_\beta^{(m)}}{\Omega} \; for \; \beta \in \{'bbox','app'\}\,, \tag{3.2}$$

3.1.4 Similarity Maps

The output of the SimNet is a set of local similarity maps having N_{max} channels in total, every channel corresponding to every target. The choice of the parameter N_{max} is a design

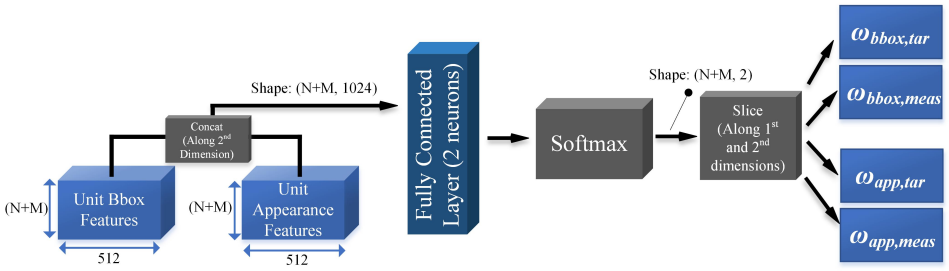


Figure 3.6: Detailed architecture of the importance branch.

choice and it directly relates to the maximum number of objects the tracker can associate at a given time instant. Initially, similarity maps are constructed by building a global map of measurement indices on the Bird's Eye View IMU/GPS frame. The resolution of the map is chosen as $0.5m$ and with a range of $80m \times 80m$ we get a map of size 160×160 pixels. Later, the measurement specific features computed by the similarity network of size 512 are placed in the respective measurement locations in the global map. Now for every target, the global map is replicated and the target-specific features of size 512 are convolved with the measurement features on the locations where the measurement is present. This corresponds to a typical convolution operation done with images but since our map is very sparse we got with selective convolutions by doing a matrix multiplication operation only at the locations where the measurements are present. This makes the algorithm much faster when compared to a full convolution with the map. The same procedure is done for other targets to build N global maps. The selective convolution operation for a typical global map channel is shown in Fig. 3.7.

The process of building these maps is done for bounding box and appearance branches giving two sets of maps. Two weight maps are built by placing the bounding box and appearance specific weights respectively, at the measurement locations. The two sets of similarity maps discussed earlier are weighted using these weight maps to produce a consolidated set of maps. Finally, the convolved output of the map will now have the similarity scores between the corresponding target with every measurement as the features are normalized to unit vectors by the network branches. Similarity maps are only built during inference. In training, the dot product of the target and measurement feature vectors is directly fed to the loss function.

The global map has dimensions $160 \times 160 \times N$. For every channel the location of the target center is identified and the global map is cropped around the target's center location up to 21 pixels on both sides which corresponds to the target's 10mx10m neighbourhood.

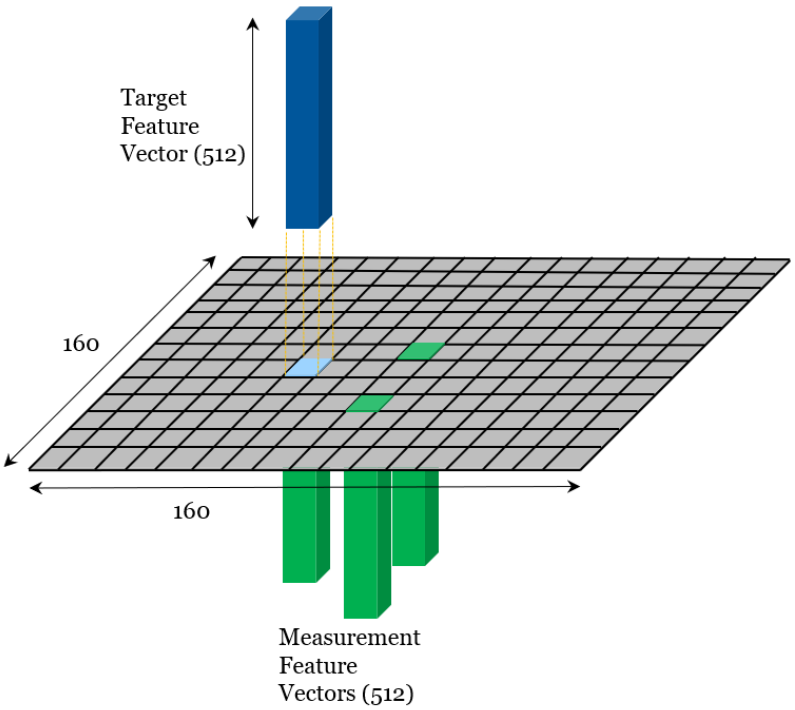


Figure 3.7: Construction of the global map. The global and local similarity maps are built only during inference.

Additional channels $N_{max} - N$ are added as dummy maps with zeros for consistency. Finally we get the localized map for every target in every channel and this is called as Local Similarity Map with dimensions $21 \times 21 \times N_{max}$.

3.1.5 SimNet Loss Function

The SimNet is a mixed architecture of trainable and non-trainable components. The bounding box branch, appearance branch, and importance branches have trainable components which need a loss function during training to measure the deviation of the prediction from the ground truth. As the problem is defined as finding the similarity score between a target and a measurement we use weighted cosine distance as the loss function. Cosine distance is widely used as a metric to measure the similarity or dissimilairty between two elements, especially in natural language processing models. Due to this reason we choose

cosine distance as the loss function. The training examples generated from the training dataset have some skewness with respect to the number of positive and negative examples (similar and dissimilar pairs). To account for this skewness we use the weighted cosine distance. The loss function for the learnable parameters Θ_1 can be given by,

$$L(\Theta) = \frac{1}{N^+} \sum_{i=1}^{N} w_{skew}^{(i)} \times w_{cost}^{(i)} \times \left(1 - y^{(i)} \times \hat{y}^{(i)}(\Theta)\right) \tag{3.3}$$

where N^+ is the number of examples with nonzero weights, $y^{(i)}$ denotes the ground truth value of the i^{th} example, i.e., $y^{(i)} \in \{-1, 1\}$. $\hat{y}^{(i)}$ is the estimated cosine similarity score computed using the cosine similarities from the two branches and their normalized importance weights as follows:

$$\hat{y}^{(i)}(\Theta) = \omega_{bbox}(\Theta)^{(i)} \times \hat{y}_{bbox}^{(i)}(\Theta) + \omega_{appear}^{(i)}(\Theta) \times \hat{y}_{appear}^{(i)}(\Theta) \tag{3.4}$$

$w_{skew}^{(i)}$ is the weight used to remove the imbalance of negative examples in the training dataset. $w_{cost}^{(i)}$ given by

$$w_{\cos t}^{(i)} = \begin{cases} -\log\left(1 - \cos^{-1}\left(\hat{y}^{(i)}(\Theta)\right)/\pi + \epsilon\right) \; if \; y^{(i)} = 1, \\ -\log\left(\cos^{-1}\left(\hat{y}^{(i)}(\Theta)\right)/\pi + \epsilon\right) \; if \; y^{(i)} = -1, \end{cases} \tag{3.5}$$

scales the loss function according to how easy or hard it is to distinguish between each pair of examples so that the training can revolve around a sparse set of the selected hard examples [53]. In 3.5 ϵ is a small constant $(1e - 10)$ that prevents taking log of zero

3.2 AssocNet - Data Association Network

The second network in the architecture does the actual data association between the targets and measurements by using the similarity scores provided by the SimNet and it is called as AssocNet. The overall architecture of the AssocNet is shown in Fig. 3.8. The input to this network is a set of local similarity maps provided by the SimNet, having dimensions $21 \times 21 \times N_{max}$. The *outputs* from the network are the predicted locations of the targets which could be translated back to the corresponding measurements thereby getting associations

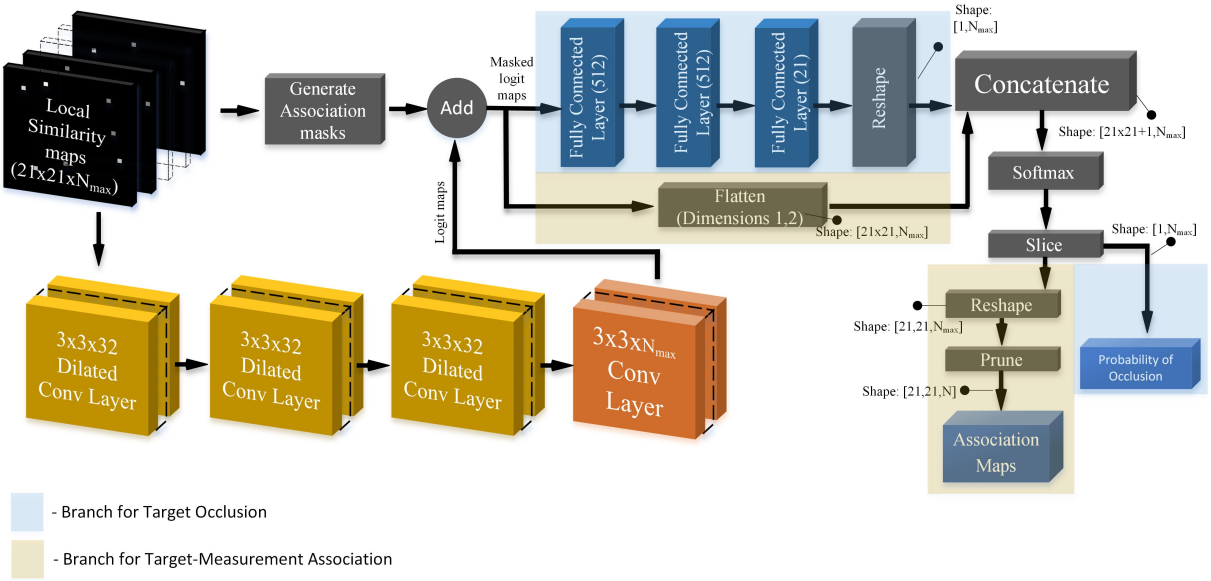


Figure 3.8: Detailed architecture of Assocnet

between targets and measurements and target-to-detection association probabilities for each existing target.

The main building blocks of the *AssocNet* are convolutional, dilated convolutional (d-Conv), and fully-connected layers, with batch-normalization and leaky-ReLU activation used in all the layers. The local similarity maps are of spatial sizes 21×21 but the locations where there are actual measurements with similarity scores are very less considering the average scenario of normal traffic in the roads when it is not extremely crowded. These sparse maps will have very small receptive fields if regular convolutions are used. Hence we use dilated convolutions which increases the receptive fields greatly [88] enabling us to learn from the sparse similarity maps.

We now discuss the data flow through *AssocNet*. The input local similarity maps are fed through a series of dilated convolutional layers with dilation factors 2,4, and 6. The neighbouring fields have slightly overlapping fields of view due to increased dilation size [70]. The convolutional layer enables interactions between these neighbouring units which effectively results in considering all the detections simultaneously while making assignments. Thus to aggregate information, we employ a 3×3 convolutional layer at the end to compute the maps of logits (the vector of non-normalized predictions).

AssocNet is to be trained to predict assignment probabilities between a target and its probable detections. Since the locations of probable detections are known in each local similarity map, there is no need to train *AssocNet* to predict assignment probabilities of other locations as zero. This reduces the training efforts for regions that are not measurement locations thereby helping in faster convergence. To implement this idea, we generate association masks for each local similarity map. In the association masks, cells of probable detections are set to zero, while the other cells are set to the smallest floating point negative number (approximating $-\infty$). This ensures the locations that do not contain the measurements remain blocked. Then the association masks are added to the map of logits obtained from the convolutional layer with $3 \times 3 \times 21$ filters (see Fig.3.8). This maintains the values of the logits computed for probable detections, but makes other logits insignificant for further computation.

After masking the maps of logits, *Assocnet* is split into two branches. The first branch has two fully connected layers having 512 neurons each and then another fully connected layer with N_{max} neurons. The output of this branch predicts the N_{max} logit values of spurious detections. These are the probabilities that the corresponding target has gone un-detected in the current frame, which also means that no association with any of the measurements could be possible. The output from this branch thus has a shape of $[1, channels]$ representing one value of probability for each of the N_{max} (we consider N_{max} instead of N for consistency and for ease of implementation) targets. The second branch flattens the maps by maintaining the channels dimension such that the shape of the second branch would be $[21 \times 21, N_{max}]$.

The first dimension corresponding to 21×21 would correspond to the logits for associating the target to all possible measurements in the spatial neighborhood of the target. Thus we solve the data association problem by modeling it as a classification problem to be solved by the network. Further we concatenate the two branches along the first dimensions such that the final shape would be $[21 \times 21 + 1, N_{max}]$. Here we add another class to the existing 21×21 classes, which will correspond to the scenario where the target has gone undetected as occlusion if it could not be classified as one among the 21×21 measurement locations. Finally a softmax is applied to this to compute the probabilities of the classes. The resulting vector is sliced such that we get the class probabilities for the 21×21 measurement locations and the 'occluded' class separately. The first vector is reshaped to $[21, 21, N_{max}]$ and the last unused channels are pruned to get the association maps. The second vector corresponding to the probabilities of occlusions are also pruned to remove the unused target channels to get the probabilities of occlusions.

From the association maps we can get back the associated measurements for every target by computing the max index locations where the probabilities are maximum.

$$(x^i_{pred}, y^i_{pred}) = \underset{x \in X^i, y \in Y^i}{\operatorname{argmax}}\ p(x, y) \qquad \forall i \in N \tag{3.6}$$

where X^i Y^i correspond to the spatial indices of the map for a given target channel i and (x^i_{pred}, y^i_{pred}) gives the predicted location of the target for the current frame which will coincide with the measurement in that location. By maintaining a dictionary of the measurement identities and their corresponding locations we could easily get back the measurement corresponding to the predicted location. Thus finally we arrive at the associations between the targets and measurements. The targets that were not associated with any of the measurements can be handled by using predictions from state estimation algorithms which will be discussed in the subsequent sections.

3.2.1 AssocNet Loss Function

Training the AssocNet is a classification problem in which the labels are the association maps showing the true associations for every target. To train the data association network we use a multi-task loss function given by,

$$L(\Theta) = l(\Theta)_{assoc} + l(\Theta)_{reg} \tag{3.7}$$

where Θ is the set parameters of the association network, $l(\Theta)_{reg}$ is the $L2$ regularization loss. $l(\Theta)_{assoc}$ is the binary cross-entropy computed for the association maps as follows:

$$\begin{aligned} q_{vec} &= q^{(t)}_{assoc}(i,j) \times \log\left(\min\left(\hat{q}^{(t)}_{assoc}(i,j;\Theta) + \epsilon, 1\right)\right) \\ p_{vec} &= p^{(t)}_{assoc}(i,j) \times \log\left(\min\left(\hat{p}^{(t)}_{assoc}(i,j;\Theta) + \epsilon, 1\right)\right) \\ l(\Theta)_{assoc} &= \sum\nolimits_{t=1}^{N} \sum\nolimits_{i,j=1}^{21+1} (-q_{vec}) + (-p_{vec}) \end{aligned} \tag{3.8}$$

where $q^{(t)}_{assoc}(i,j) = 1 - p^{(t)}_{assoc}(i,j)$ and ϵ is the margin used to ignore negligible errors in the predicted probabilities $\hat{p}^{(t)}_{assoc}(i,j;\Theta)$. The upper limit of i and j in the inner summation in 3.8 is $21+1$ as we need to account for the additional class which corresponds to the probability of occlusion, in addition to the 21 classes corresponding to the actual measurement locations.

3.2.2 Visualization of the maps

To visualize the global and local similarity maps belonging to Simnet and the predictions of the AssocNet, we consider an example from the training video 17. We have two frames 90 and 91 at two consecutive time instants, which are shown in Fig. 3.10. We have four targets from frame 90 and four measurements in the frame 91. For simplicity, we only consider the target belonging to track 0. Fig.3.9a shows the global similarity map generated for the target track 0. We could see that the most probable measurement that is similar to the target is given a higher similarity score. These global maps are cropped around the respective targets to arrive at local similarity maps. The local similarity map and the corresponding output prediction by AssocNet are shown in Fig. 3.9b.

Figure 3.10: Two frames 90, and 91 from the training video 17.

3.3 Track Management

Let us assume we have a set T of all the trajectories t_i. At any time instant $\tau = k$, we have the set of measurements M of the form m^k. Successful associations are denoted as (t_i^k, m_j^k), occlusions as $(t_i^k, None)$, and track birth as $(None, m_j^k)$. The track management module consists of a Kalman filter for state estimation and a probability of existence for maintaining the tracks. We discuss these two in detail in the upcoming subsections.

3.3.1 Kalman Filter

The output of the detector could contain noisy measurements which are typically filtered by state estimation algorithms. Furthermore, the undetected targets which are occluded could re-appear after a few frames. There should be a good motion prediction mechanism which

would help in re-identifying the missed target, which will improve the overall tracking accuracy and reduce fragmentation. To address these we use a linear Kalman filter for motion prediction and state estimation. We track the 3D positional coordinates of the object as the state in the filter, and the velocities in the corresponding dimensions are derived from the filter by using kinematic equations as motion model. The state of the filter X is given by,

$$[x, y, z, \dot{x}, \dot{y}, \dot{z}] \tag{3.9}$$

where x, y, z are the coordinates of the center of the object being tracked in IMU-GPS coordinates of the first frame of the ego car (referred as IMU_0) which are also called as measurements and $\dot{x}$ $\dot{y}$ and $\dot{z}$ are the corresponding velocities. To remove the ego motion we transform the positions from camera coordinates to IMU_0 using transformation matrices and GPS data [25].

Thus the state dimensions are 6 and measurement dimensions are 3. The state transition matrix is given by,

$$F = \begin{bmatrix} 1 & 0 & 0 & \delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.10}$$

The term δt in the last three columns help in updating the state according to the Newtoninan equation $x_t = v\delta t + x_{t-1}$ (here x is the position and v is the velocity). Using the F matrix this generalizes for all dimensions of the state as $X_t = FX_{t-1}$. The velocity in the state is initialized by following a two-point initiation algorithm [55]. The velocity is set to 0 at time $t = 0$. At time $t = 1$ the velocity is updated in the filter as $v = x_t - x_{t-1}$.

The H matrix which represents the measurement function that is used to transform the state space to the measurement space is initialized as an identity matrix. The R matrix representing the measurement noise covariance is derived from the MSE values of the position estimates from the detector, using ground truth data from the training set. This represents the noise of the object detector.

$$R = \begin{bmatrix} 0.051589 & 0 & 0 \\ 0 & 0.013226 & 0 \\ 0 & 0 & 0.010612 \end{bmatrix} \tag{3.11}$$

The Q matrix which represents the covariance of the noise of the process is set as discrete white noise [46] with a covariance factor σ^2 which is set to 1. The Q matrix is computed according to [46] as follows,

$$Q = \begin{bmatrix} \delta t^4/4 & \delta t^3/2 & \delta t^2/2 \\ \delta t^3/2 & \delta t^2 & \delta t \\ \delta t^2/2 & \delta t & 1 \end{bmatrix} \sigma^2 \tag{3.12}$$

The P matrix of the filter which corresponds to the state error covariance is computed [55] using the R matrix as follows (subscripts indicate the time instants),

$$P_{2|2} = \begin{bmatrix} R_2 & \frac{1}{\delta t} R_2 \\ \frac{1}{\delta t} R_2 & \frac{1}{\delta t^2}(R_1 + R_2) \end{bmatrix} \tag{3.13}$$

3.3.2 Probability of Existence

The track existence probability P_e^i helps in pruning the tracks with Bayesian estimation. During the prediction step we set a prior as

$$P_{e_i}^k = P_{e_i}^{k-1} \times P_{surv} \quad \forall i \in T \tag{3.14}$$

where P_{surv} is the probability of survival which is set as 0.60. For every successful association we compute P_e^i [61] as

$$P_{e_i}^k = \frac{1 - \Delta_i^k}{1 - P_{e_i}^{k-1} \Delta_i^k} \times P_{e_i}^{k-1} \quad where \quad \Delta_i^k = \omega_i \times \left[1 - \frac{\mathcal{L}_i}{\delta_i}\right] \quad \forall i \in T \tag{3.15}$$

Here ω_i is the detection score of the measurement associated with t_i, $\mathcal{L}_i$ is the Gaussian likelihood of the measurement. δ_i is the clutter intensity which is modeled as a Poisson process [61] as

$$\delta_i = \lambda \times c(m_j^k)$$

where λ is the poisson clutter rate of the detector estimated using the training data and $c(m_j^k)$ is the spatial distribution modeled as uniform density in the perception volume of the detector ($80\,\text{m} \times 70\,\text{m} \times 8\,\text{m}$) and is given by

$$\frac{1}{\left[(80 - l_i + 1) \times (70 - w_i + 1) \times (8 - h_i + 1)\right]}$$

where l_i, w_i, h_i are the length, width, and height respectively of the measurement.

3.4 Experiments

The following subsections explain in detail about the datasets used in training the networks, the hyper-parameters used in training, the experiments performed and the quantitative and qualitative results obtained.

3.4.1 Dataset

We used the KITTI Tracking benchmark dataset for training and evaluation of our approach. The KITTI Tracking dataset consists of 21 training sequences and 29 test sequences. As the training sequences have different levels of difficulty, occlusion, and clutter we split the 20% of every training sequence for validation. Thus the first 80% of a tracking sequence would be used during training and the last 20% of the sequence would be used during validation. This way, training and validation datasets are not skewed and have seen all the scenarios in the dataset which occur in specific video sequences. For training SimNet, we construct a training dataset from the training sequences by generating positive and negative examples in consecutive frames and odd and even frames using ground truth information. Geometric transformations (translation, rotation, and scaling) are applied to the ground-truth bounding box parameters to model partial occlusion and detector noise. This gives a large training set in which the ratio of negatives to positives is approximately $18 : 25$.

We trained the object detector using a combined dataset consisting of the KITTI 3D object detection dataset and the 80% split of the KITTI training dataset mentioned earlier, after pre-training on a synthetic dataset [36]. Pretraining with the synthetic dataset increases the detection accuracy by about 3%. We choose the best checkpoint for the object detector based on the best 3D object detection AP (Average Precision) on the validation set of the combined dataset. We plot the P-R curve as shown in Fig. 3.11 and find the best threshold corresponding to the maximum F1 score as 0.28.

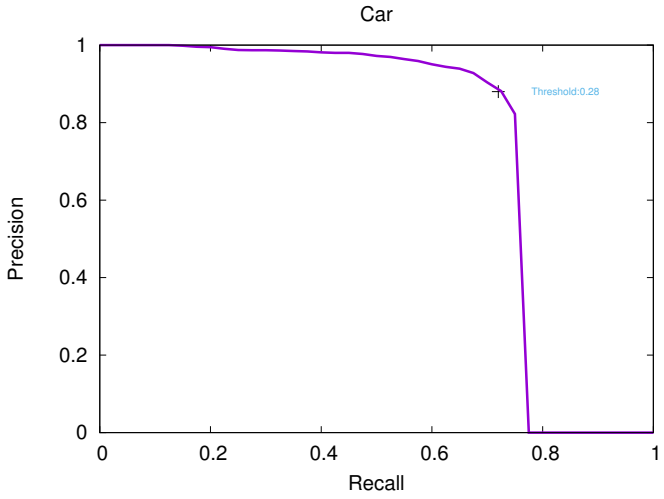


Figure 3.11: PR curve of the object detector (AVOD) on the combined dataset for the car class. The best threshold 0.28 corresponds to the point where F1 score is maximum.

3.4.2 Training

We build a simnet dataset to train the Simnet by constructing mini batches of batch size 128. Each mini batch has components as shown in Table 3.1.

The AssocNet is designed to take a similarity map as input and perform the data association. As the AssocNet operates on a pair of frames (frame corresponding to time τ and $\tau-1$) the inputs for training are prepared by taking the actual ground truth detections in the training set at times τ and $\tau-1$. We used a batch size of 1 for AssocNet. The local similarity maps for training are generated by running the trained SimNet on the training dataset. The label maps for the inputs are generated by mapping the right measurement that is the ground truth association for a given sample target on a localized map and concatenating such maps for all the targets in the scene. This gives rise to the label which is similar in shape to the local similarity map. The structure of the AssocNet dataset is shown in Table 3.2.

To optimize the parameters using the loss function (3.3) we use Adam optimizer and exponentially-decaying learning rate [41]. The learning rate is initially set to $1e-5$ and then decreased every 100 epochs with a base of 0.95. AssocNet also uses Adam optimizer to optimize the parameters using the loss function defined in equation 3.7. The learning

rate for AssocNet is initially set to $1e-6$ and then decreased every 20 epochs with a base of 0.95.

3.4.3 Evaluation Metrics

The accuracy in SimNet is measured (3.16) by counting the number of correct predictions (similar or dissimilar) by comparing them to the ground truth. We obtain a training accuracy of 90.5% and validation accuracy of 91.3%.

$$accuracy_{simnet} = \frac{\sum_{i=1}^{N}(y^i == \hat{y}^i)}{N} \qquad y \in \{0, 1\} \tag{3.16}$$

The AssocNet predicts the (x,y) locations of the probable measurement in the local map of every target. Hence its accuracy is given by considering all the correct predictions in all the target channels as shown in 3.17. We obtain a training accuracy of 99.71 % and validation accuracy of 99.78 %. We compare these spatial predictions with the ground truth according to the following equation

$$accuracy_{assocnet} = \frac{\sum_{i=1}^{N}(x^i == \hat{x}^i) \wedge (y^i == \hat{y}^i)}{N} \qquad x, y \in [-10, 10] \tag{3.17}$$

where x^i and y^i are the spatial predictions of associated measurement locations for the target i.

These metrics are used to evaluate the performance of the individual networks. To evaluate the overall performance of tracking, we use the popular CLEAR MOT metrics [8]. Multiple Object Tracking Accuracy (MOTA) gives us an estimate of the tracker's overall performance. However, this is dependent on the performance of the object detector. Hence, we also look at tracking specific metrics like Mostly Tracked (MT), Mostly Lost (ML), ID Switches (IDS) and fragmentation (FRAG), which evaluate the efficiency of the tracker in assigning the right IDs with reduced switches or fragmentation in the tracks. Mostly Tracked gives an estimate of the proportion of tracks that are tracked successfully for more than 80% of their lifetime. Mostly Lost gives an estimate of the proportion of tracks that are tracked for less than 20% of their lifetime. When the IDs of two different tracks are swapped we call this as an ID switch. If a track gets a new ID in its lifetime we call this as a Fragmentation.

3.4.4 Ablation Study

We do an ablation study to evaluate the components in our approach by comparing them with traditional approaches. Firstly, we study the impact of the similarity network. In Table 3.3, Euclidean and Manhattan denote the baseline distances modeled with the 3D position estimates. Bhattacharyya and ChiSquare metrics are built from the image histograms of the cropped targets and detections to study the image-only configuration. SimNet and AssocNet denote our similarity and Association networks respectively. From Table 3.3, we could infer that conventional similarity approaches were not able to achieve comparable accuracy (MOTA) as the features involved in the computation of the similarity scores were not robust. We also study the impact of our association network by replacing it with a baseline Hungarian approach. Again, we could observe that the baseline approaches like Hungarian couldn't fare better than ours. Finally, we study the relative importance of the individual branches in the Simnet by disabling either of the inputs. By losing either of the inputs the overall MOTA is seen to decrease and at the same time it also leads to more ID switches and fragmentation.

3.4.5 Experiments with Tracking Parameters

We also perform experiments to fine tune the parameters for the Kalman filter and the track existence. From Table 3.4 we could see that the results on the validation set when the existence threshold of the tracks is set to 0.30, 0.40 and 0.50 and the survivability factor values set at 0.50, 0.60 and 0.70 by setting everything else constant and the best results are obtained while setting them to 0.40 and 0.60 respectively. The probability of existence update which is performed for every successful track association relies on the Gaussian likelihood which in turn depends on the Kalman filter prediction. During the initial few frames of every track the prediction would not have converged and this leads to lower likelihood and premature termination of tracks. To avoid this we delay the update by doing a probability of existence reversal until the track age is 5 frames.

3.4.6 Kitti Benchmark Results

We evaluate our approach on the test sequences on the KITTI evaluation server for the 'Car' class. The results are presented in Table 3.5.

Due to the challenging nature of online tracking approach and to do a fair comparison, we only consider published online tracking approaches for our comparison. We achieve

competitive results with respect to the state of the art in online tracking with improved MOTP which is better than most of the online methods. Our Mostly Tracked and Mostly Lost (MT & ML) values are also competitive which show the effectiveness of our data association approach. Further, our approach gives inferences in 3D and KITTI evaluations are done in 2D, which is not completely representative of our approach. It should also be noted that none of these approaches use deep learning for data association. On the other side, we have used a simple Kalman filter for state estimation and motion prediction which could potentially be improved by better tuning of parameters or trying out more sophisticated approaches for track management.

3.4.7 Qualitative Evaluation

We perform a qualitative evaluation by running our tracker on the KITTI tracking validation and testing sequences. We analyze different scenarios including occlusions, clutter, parked vehicles and false negatives from the detector. Fig. 3.14 shows an example from sequence 0 in the test set. Different tracks representing the vehicles are color coded and the track IDs are displayed for reference. The tracker is able to perform well in spite of the clutter due to the closely parked cars. In Fig. 3.12 we see an example from test sequence 17 in which the false negative by the detector is overcome with the help of the prediction of the tracker. These examples show the robustness of the tracker and its ability to perform better in scenarios of missed detections. There were also some cases where the data association fails and as a result ID switching and fragmentation happen. In Fig. 3.13 the track 38 was previously assigned to a nearby car but after an occlusion in the detection ID switching happens. This could be due to the low-lit conditions of the two cars.

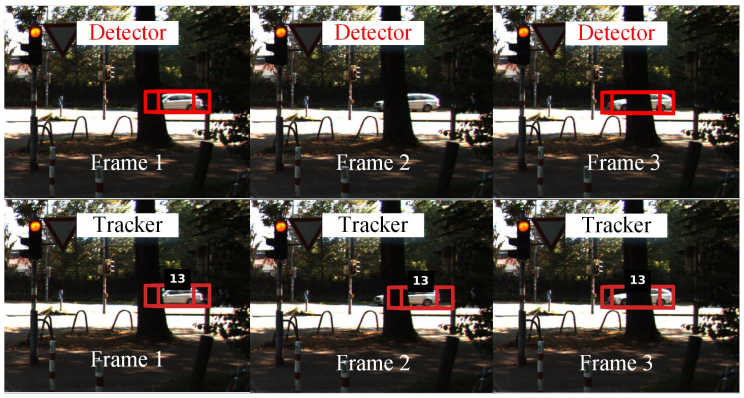


Figure 3.12: Qualitative Evaluation - In this example (video 17 in test set) the detection was missed by the detector and reappears in the next frame. But the tracker was able to successfully maintain the track

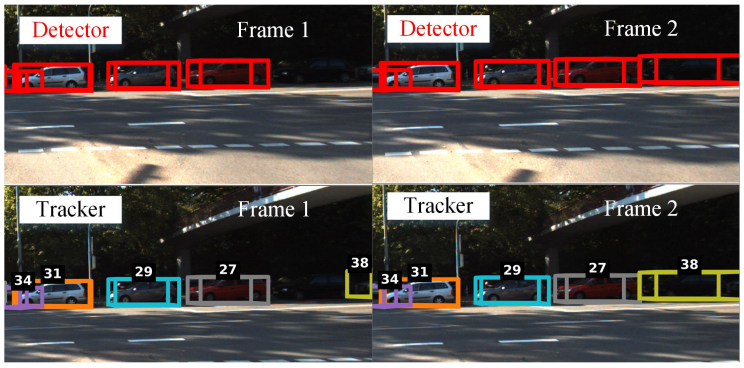


Figure 3.13: Qualitative Evaluation - An example from video 15 in test set where ID switching occurs for Track 38 due to low-lit conditions

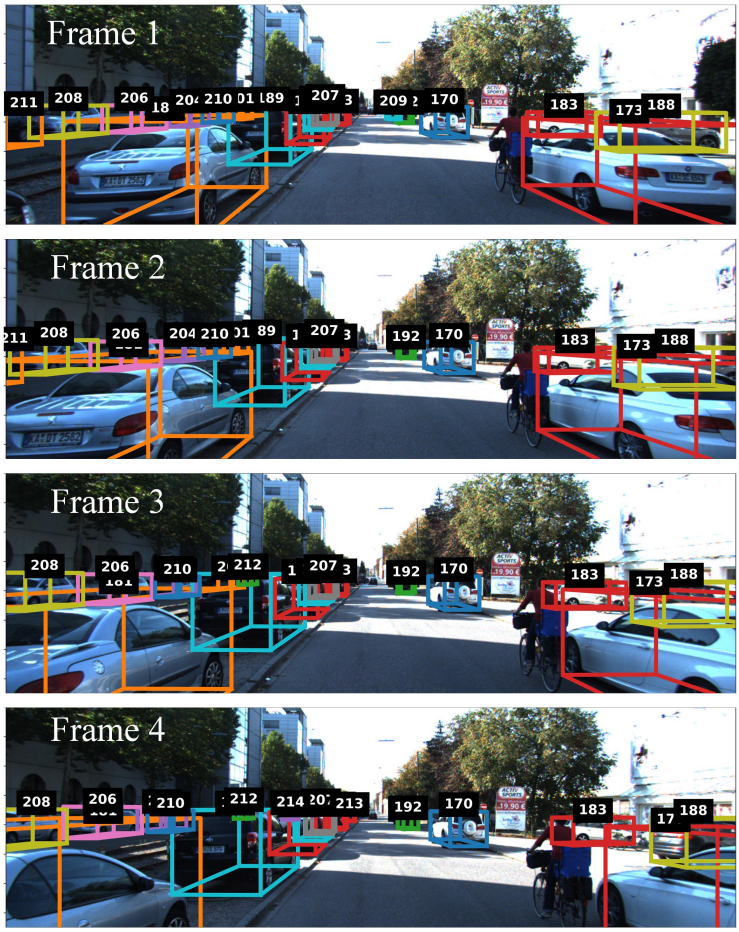
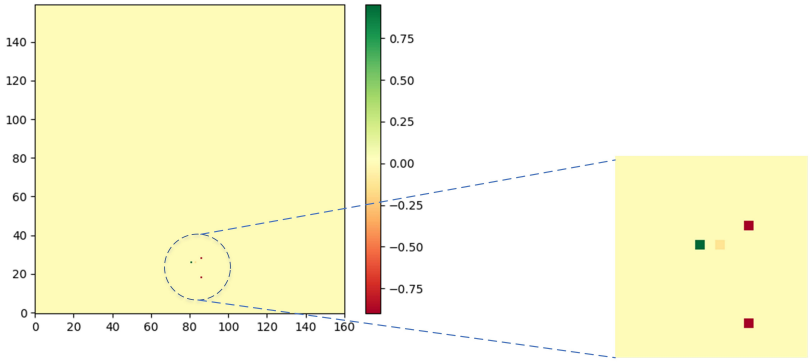
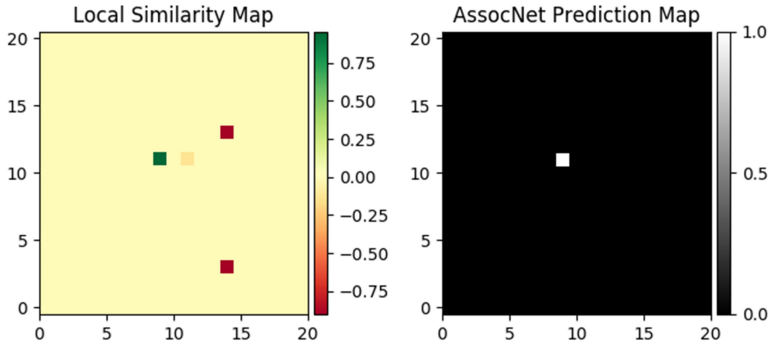


Figure 3.14: Qualitative Evaluation - An example from video 14 in test set where the tracker performs well in a cluttered scene with parked cars.



(a) Global similarity map



(b) Local similarity map and AssocNet prediction

Figure 3.9: (a) shows the global similarity map and (b) shows the Local Similarity map and AssocNet's prediction corresponding to target track 0.

Algorithm 1: Tracker Algorithm

```
P_surv = 0.60 ;                                   // Probability of Survival
θ_ex = 0.40 ;                                     // Existence threshold
while true do
    Get measurements m^k at time τ = k
    if k = 0 then
        foreach m_i^0 do
            Create new track i
        end
    else
        PredictTracks :
            Perform Kalman Filter Prediction;
            Compute prior P_{e_i}^k;

        DataAssociation :
            Data Association for t^k and m^k;

        UpdateTracks :
            Perform Kalman Filter Update;
            Update P_{e_i}^k;
            ∀(t_i^k, m_j^k) Update track i with m_j^k
            ∀(t_i^k, None) Propagate predicted t_i^k to τ = k + 1
            ∀(None, m_j^k) Create a new track;

         ∀i ∈ T if P_{e_i}^k < θ_ex then
            Prune i
        end
    end
end
```

Input	**Shape**
Targets' Bounding boxes	[128,1,7,1]
Measurements' Bounding boxes	[128,1,7,1]
Targets' appearance features	[128,7,7,640]
Measurements' appearance features	[128,7,7,640]
Labels	[128,1]

Table 3.1: Inputs to SimNet

Input	**Shape**
Local Similarity Maps	[1,21,21,21]
Number of Targets	[1,1]
Label Maps	[1,21,21,21]

Table 3.2: Inputs to AssocNet

Method	MOTA ↑	MOTP ↑	MT ↑	PT ↑	ML ↓	IDS ↓	FRAG ↓
Euclidean+*AssocNet*	56.16 %	84.84 %	72.22 %	18.51 %	9.25 %	269	320
Manhattan+*AssocNet*	56.75 %	84.83 %	**73.14 %**	17.59 %	9.25 %	265	319
Bhattacharyya+*AssocNet*	56.69 %	84.81 %	72.22 %	18.51 %	9.25 %	256	307
ChiSquare+*AssocNet*	57.17 %	84.81 %	**73.14 %**	18.51 %	**8.33 %**	262	311
SimNet+Hungarian	74.59 %	**84.92 %**	65.74 %	**23.14 %**	11.11 %	26	93
SimNet ImgOnly+AssocNet	74.30 %	84.75 %	73.14 %	17.59 %	9.25 %	29	82
SimNet BboxOnly+AssocNet	75.51 %	84.74 %	72.22 %	18.51 %	9.25 %	15	70
SimNet+AssocNet	**76.52 %**	84.81 %	**73.14 %**	17.59 %	9.25 %	**1**	**54**

(↑ denotes higher values are better. ↓ denotes lower values are better)

Table 3.3: Ablation study on KITTI validation set for 'Car' class

Parameter	MOTA ↑	MOTP ↑	MT ↑	ML ↓	IDS ↓	FRAG ↓
$\theta_{ex} = 0.30$	76.34%	84.60%	75%	8.33 %	2	45
$\theta_{ex} = 0.50$	76.46 %	84.82 %	73.14 %	10.18 %	1	58
$P_{surv} = 0.50$	76.46 %	84.82 %	73.14 %	10.18 %	1	58
$P_{surv} = 0.70$	76.34 %	84.61 %	75 %	8.33 %	2	44
$\sigma^2 = 0.1$	75.87 %	84.78 %	72.22 %	9.25 %	4	76
$\sigma^2 = 10$	76.49 %	84.80 %	72.22 %	9.25 %	4	56
$\sigma^2 = 50$	76.17 %	84.83 %	72.22 %	9.25 %	5	58
Optimal ($\theta_{ex} = 0.40$,$P_{surv} = 0.60$, $\sigma^2 = 1$)	**76.52 %**	84.81 %	73.14 %	9.25 %	**1**	54

(↑ denotes higher values are better. ↓ denotes lower values are better)

Table 3.4: Experiments with Tracking Parameters

Method	**MOTA ↑**	**MOTP ↑**	**MT ↑**	**ML ↓**	**IDS ↓**	**FRAG ↓**
MOTBeyondPixels [73]	**84.24** %	**85.73** %	**73.23** %	**2.77** %	468	944
JCSTD [87]	80.57 %	81.81 %	56.77 %	7.38 %	**61**	643
3D-CNN/PMBM [72]	80.39 %	81.26 %	62.77 %	6.15 %	121	613
extraCK [29]	79.99 %	82.46 %	62.15 %	5.54 %	343	938
MDP [86]	76.59 %	82.10 %	52.15 %	13.38 %	130	**387**
FANTrack (Ours)	77.72 %	82.32 %	62.61 %	8.76 %	150	812

(↑ denotes higher values are better. ↓ denotes lower values are better)

Table 3.5: Results on Kitti Test set for 'Car' class

Chapter 4

Implementation

This chapter describes the implementation of FANTrack which is a full-fledged tracker implemented as part of the software stack on the level 3 autonomous driving research platform developed at the University of Waterloo called "Autonomoose". We describe the system architecture, dependencies and working of the tracker as a ROS Node. The source code of the implementation is available at [2].

4.1 Architecture

The FanTrack module requires the camera images, 3D bounding boxes from the object detector, and static and dynamic transforms. The camera images are available as ros topics from the cameras. We use the Front camera for straight driving and three cameras (LeftFront, Front and RightFront) for scenarios involving intersections. We get the transforms for `camera_F` to `imu_F` from the calibration publisher which publishes the static transforms. We get the `odom` to `imu_F` and `imu_F` transforms from the localizer which provides the dynamic transforms involving `odom` frame of reference. We use `imu_F` for the data association step as this frame of reference is vehicle body frame. We perform the filter updates in the `odom` frame as this serves as a continuous fixed frame of reference which doesn't move with the vehicle for short durations. The tracker publishes the list of tracks as a ROS topic `\obj_tracked`. A component diagram which shows the interfacing of the tracker with its dependencies is shown in Fig. 4.1.

To implement the tracker as part of the autonomous driving software stack we consider two architectural choices based on how the tracker interacts with the object detector.

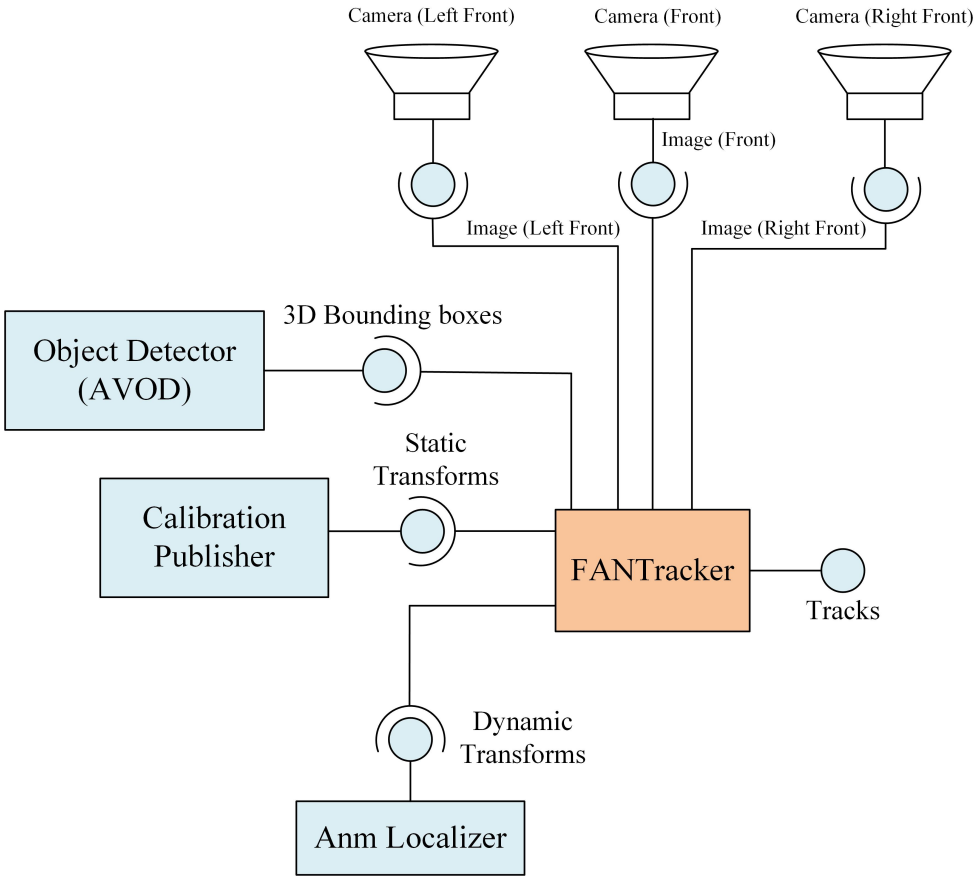


Figure 4.1: A component diagram of FANTracker with dependencies, required and provided interfaces.

4.1.1 Coupled approach

In this approach, the tracker is a submodule of the object detector (AVOD). The tracker will publish its own obj_tracked topic but it is invoked as part of the AVOD's callback function. This is also regarded as a synchronous approach with respect to the object detection and tracking as the object detector has to wait until the tracks are assigned to the detected objects. The coupled approach is represented as a sequence diagram in Fig. 4.2. In this approach we extract the camera image from the one that is fed to the object detector and the bounding boxes also have the same timestamp.

The advantage of this approach is that we don't have to explicitly synchronize the

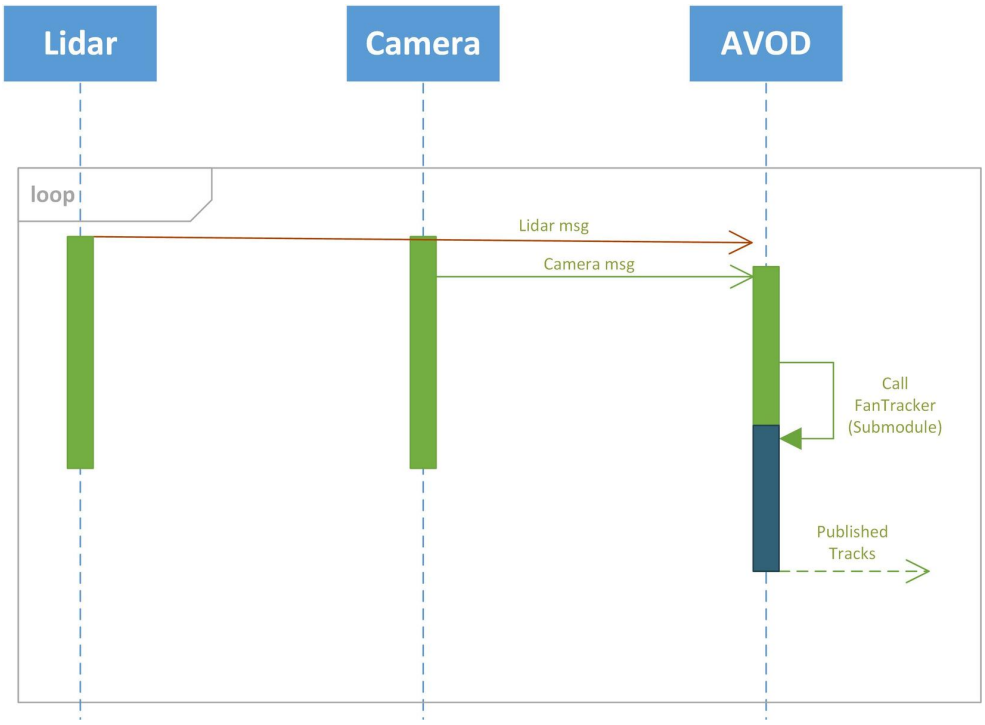


Figure 4.2: Synchronous approach in which the FANTracker is a submodule of the object detector

camera image and the bounding box inputs as the entire process is synchronous. This will avoid missed detections due to synchronization issues. But the disadvantage of this approach is that the object detector and the tracker are sequential in nature and this prevents either of them in running in parallel with the other. This leads to under utilization of resources and delays the inference times which has a cascading effect on the downstream systems. Furthermore, tight coupling of components is generally not encouraged in software systems.

4.1.2 Decoupled approach

In this approach, the tracker is designed as an independent ROS node which accepts inputs from the camera and the object detector. This can also be regarded as an asynchronous approach as the object detector doesn't wait till the tracking is done and the final tracked

object list is published. The sequence diagram of the decoupled approach is shown in Fig. 4.2.

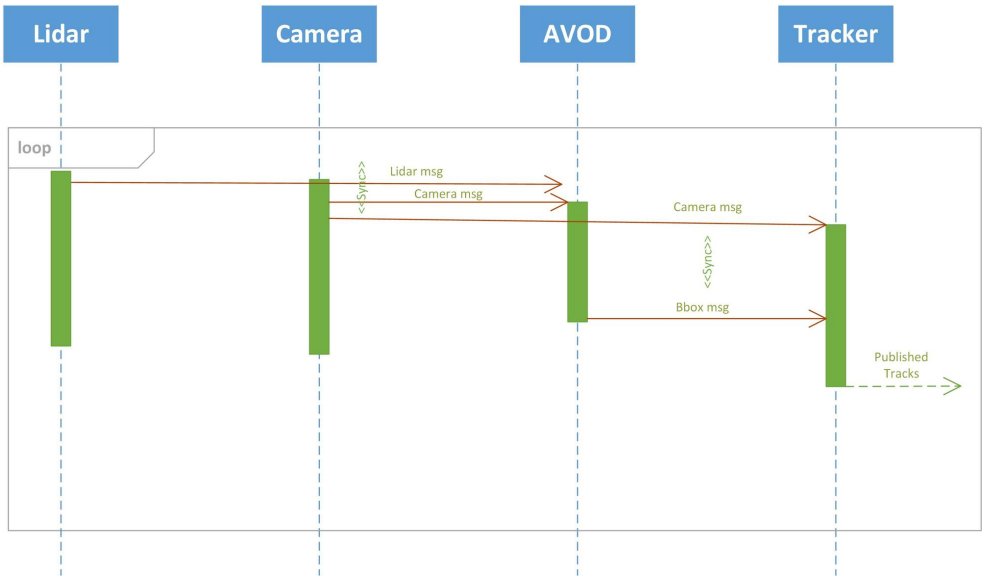


Figure 4.3: Asynchronous approach in which the FANTracker is an independent module

The major advantage of this approach is that the components can run in parallel and they don't have to wait for the other to finish processing. This increases resource utilization which is the main drawback of the previous approach. By comparing both the approaches we can conclude that the decoupled approach has better advantages as it increases resource utilization and has less coupling.

To overcome the disadvantages posed by the varying timestamps of the camera and bounding box inputs, we utilize the concept of time synchronization that helps in synchronizing the ROS messages. We use the `ApproximateTimeSynchronizer` which is part of the `message\filters` ROS library.

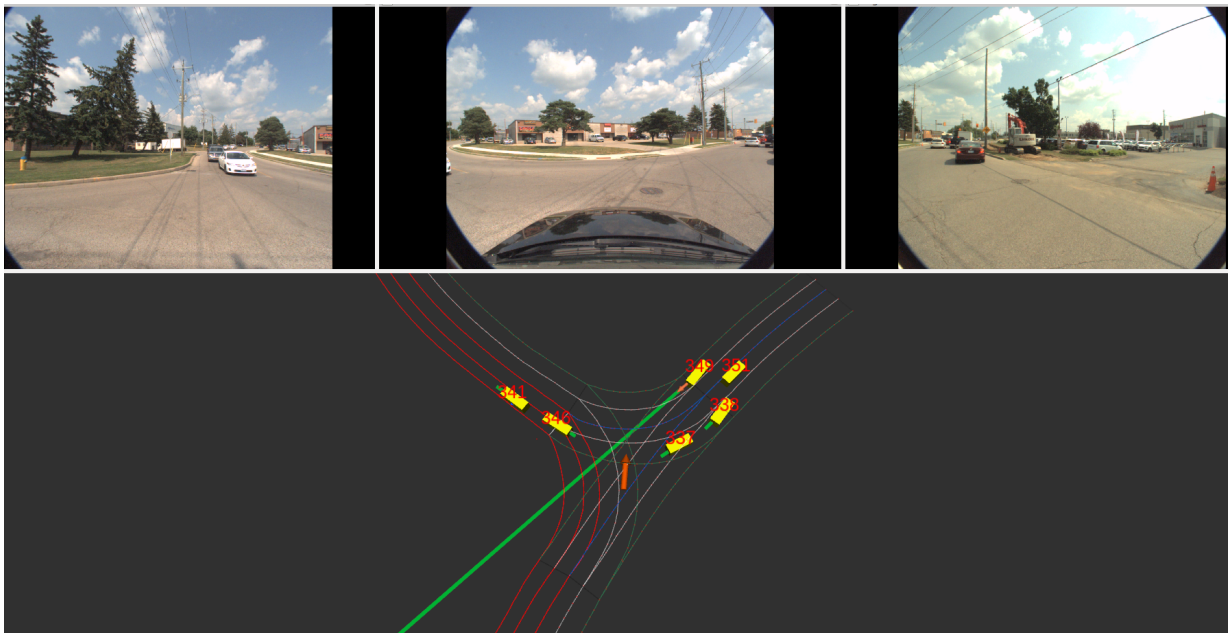
(a) Frame 1

(b) Frame 2

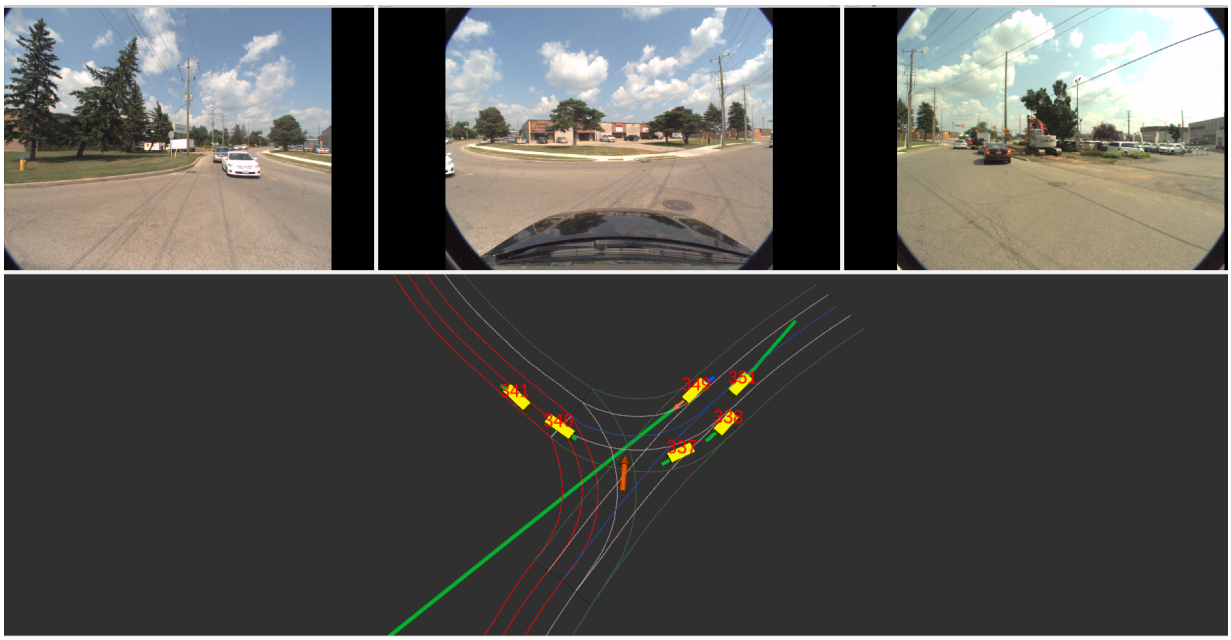
(c) Frame 3

(d) Frame 4

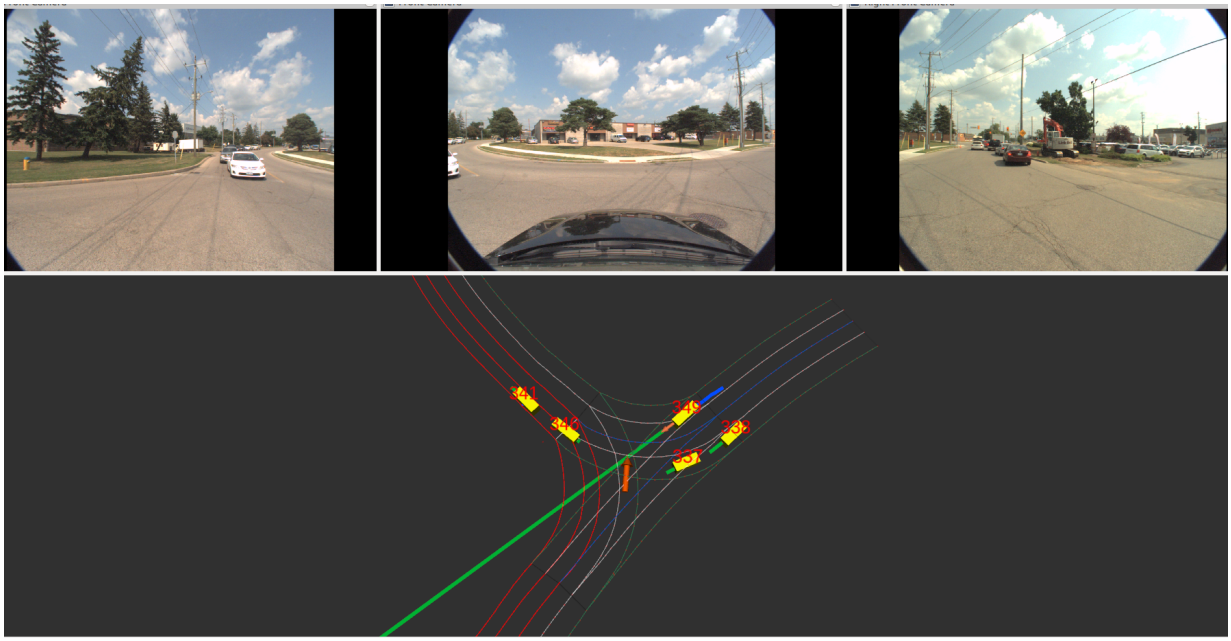
Figure 4.4: Visualization of the FANTracker running in Autonomoose. Normal driving scenario with only Front Camera)



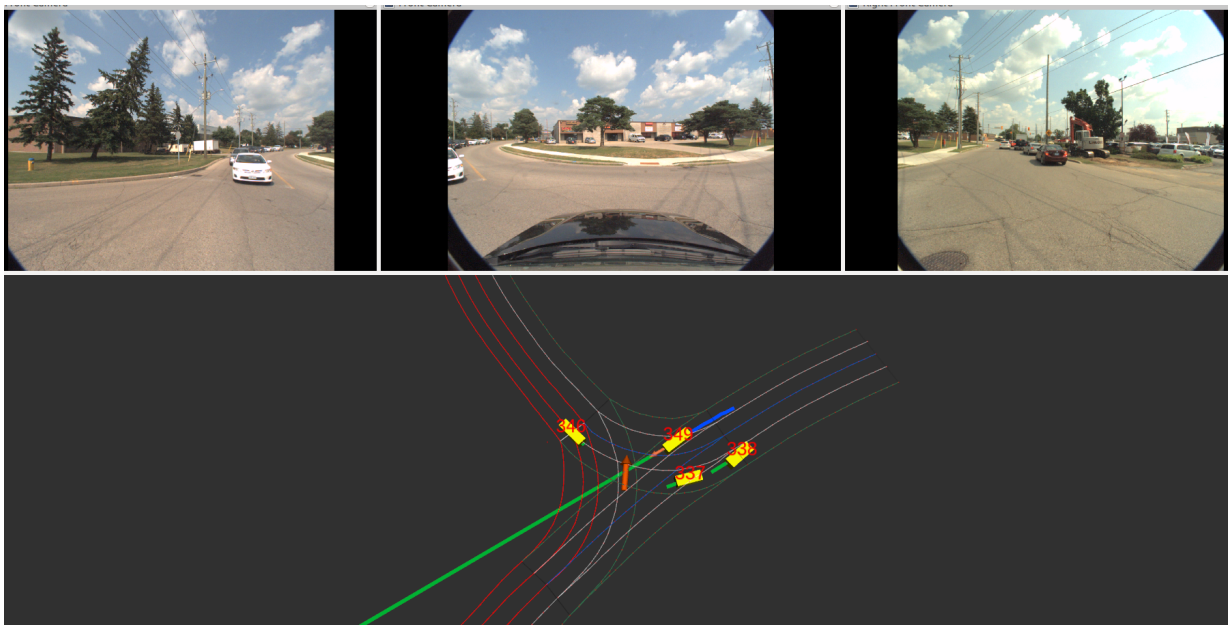
(a) Frame 1



(b) Frame 2



(c) Frame 3



(d) Frame 4

Figure 4.5: Visualization of the FANTracker running in Autonomoose. Intersection scenario with three cameras

Chapter 5

Conclusion

This work introduced FANTrack, a 3D online multi-object tracking approach for autonomous driving scenarios. FANTrack differs from the other online tracking approaches in its ability to efficiently fuse multi-modal inputs from different sensors to compute similarity scores and a learning based data association implemented as inference in CNN. Experiments on Kitti tracking benchmark showed competitive results in online tracking and state of the art results considering published 3D online tracking strategies. FANTracker is implemented as a ROS node and tested in our autonomous vehicle for real time performance.

5.1 Limitations

5.1.1 Performance limitations

Though the FANTracker's frame-wise data association is very good, the overall tracking performance is limited by the performance of the object detector as it relies on the detections from a detector. Missing detections for more frames and frequent false positives from the object detector which are beyond the limits of the filter would degrade the overall tracking accuracy.

5.1.2 Overfitting

As the FANTrack involves deep neural networks in the similarity and association architectures it needs a lot of training data. If the networks are not trained with a sufficient

amount of data the models are susceptible to overfitting which would reduce the performance during inference. Furthermore, we need labeled data to train the networks and getting the datasets labelled is often considered as a time consuming effort.

5.2 Future Work

5.2.1 Combined Architectures

We train the Simnet and AssocNet as two separate networks and use them separately during inference. We could combine these two networks into a single architecture and training can be done in tensorflow by removing the SimNet loss function and connecting the similarity maps directly to the AssocNet placeholders. Further research is required in designing a combined loss function as the process of making the similarity maps to predict the similarity values and the assocnet's responsibility of classification should be factored into the new loss function. Furthermore, the object detection framework could be combined with the data association by feeding in the detections from the previous frame. This would lead to an end-to-end detection and tracking framework.

5.2.2 Local Map size

The resolution of the local similarity map represents the extent to which the network should look for the measurement to associate the target. This could be related to the speed of the vehicles and further research is possible in altering the size of the map size or dynamically varying it according to some input factors like the type of road the car is currently driving on.

5.2.3 Prediction and Track Maintenance

We use Kalman filters for prediction and a probability of existence formulation to decide when to terminate the tracks. Instead, sequence-based recurrent neural network models like LSTMs could be used to predict and maintain the trajectories.

References

[1] Amit Adam, Ehud Rivlin, and Ilan Shimshoni. Robust fragments-based tracking using the integral histogram. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 1, pages 798–805. IEEE, 2006.

[2] Venkateshwaran Balasubramanian. Fantracker (ros integration), Jul 2019.

[3] Y Bar-Shalom, S Blackman, and RJ Fitzgerald. The dimensionless score function for measurement to track association. *IEEE Trans Aerosp Electron Syst*, 41(1):392–400, 2007.

[4] Y. Bar-Shalom, F. Daum, and J. Huang. The probabilistic data association filter. *IEEE Control Systems Magazine*, 29(6):82–100, Dec 2009.

[5] Erkan Baser, Venkateshwaran Balasubramanian, Prarthana Bhattacharyya, and Krzysztof Czarnecki. Fantrack: 3d multi-object tracking with feature association network. *arXiv preprint arXiv:1905.02843*, 2019.

[6] Sean Bell and Kavita Bala. Learning visual similarity for product design with convolutional neural networks. *ACM Transactions on Graphics (TOG)*, 34(4):98, 2015.

[7] Jérôme Berclaz, Franois Fleuret, Engin Türetken, and Pascal Fua. Multiple object tracking using k-shortest paths optimization. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33:1806–1819, 2011.

[8] Keni Bernardin and Rainer Stiefelhagen. Evaluating multiple object tracking performance: the clear mot metrics. *Journal on Image and Video Processing*, 2008:1, 2008.

[9] Christopher M Bishop. Regularization and complexity control in feed-forward networks. In *Proceedings International Conference on Artificial Neural Networks ICANN*, volume 95, pages 141–148, 1995.

[10] Henk AP Blom and Edwin A Bloem. Probabilistic data association avoiding track coalescence. *IEEE Transactions on Automatic Control*, 45(2):247–259, 2000.

[11] William Brendel, Mohamed R. Amer, and Sinisa Todorovic. Multiobject tracking as maximum weight independent set. *CVPR 2011*, pages 1273–1280, 2011.

[12] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[13] Visesh Chari, Simon Lacoste-Julien, Ivan Laptev, and Josef Sivic. On pairwise costs for network flow multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5537–5545, 2015.

[14] Sumit Chopra, Raia Hadsell, Yann LeCun, et al. Learning a similarity metric discriminatively, with application to face verification. In *CVPR (1)*, pages 539–546, 2005.

[15] Alexis B Cook. Global average pooling layers for object localization. https://alexisbcook.github.io/2017/global-average-pooling-layers-for-object-localization/.

[16] Mauro Dell'Amico and Paolo Toth. Algorithms and codes for dense assignment problems: the state of the art. *Discrete Applied Mathematics*, 100(1-2):17–48, 2000.

[17] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee, 2009.

[18] Alexey Dosovitskiy, Philipp Fischer, Eddy Ilg, Philip Hausser, Caner Hazirbas, Vladimir Golkov, Patrick Van Der Smagt, Daniel Cremers, and Thomas Brox. Flownet: Learning optical flow with convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 2758–2766, 2015.

[19] Oliver E Drummond. Best hypothesis target tracking and sensor fusion. In *Signal and Data Processing of Small Targets 1999*, volume 3809, pages 586–601. International Society for Optics and Photonics, 1999.

[20] Volker Eiselein, Daniel Arp, Michael Pätzold, and Thomas Sikora. Real-time multi-human tracking using a probability hypothesis density filter and multiple detectors. In *2012 IEEE Ninth International Conference on Advanced Video and Signal-Based Surveillance*, pages 325–330. IEEE, 2012.

[21] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Detect to track and track to detect. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3038–3046, 2017.

[22] Thomas Fortmann, Yaakov Bar-Shalom, and Molly Scheffe. Sonar tracking of multiple targets using joint probabilistic data association. *IEEE journal of Oceanic Engineering*, 8(3):173–184, 1983.

[23] Davi Frossard and Raquel Urtasun. End-to-end learning of multi-sensor 3d tracking by detection. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 635–642. IEEE, 2018.

[24] Kunihiko Fukushima. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological cybernetics*, 36(4):193–202, 1980.

[25] Andreas Geiger, Philip Lenz, Christoph Stiller, and Raquel Urtasun. Vision meets robotics: The kitti dataset. *The International Journal of Robotics Research*, 32(11):1231–1237, 2013.

[26] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[27] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[28] Helmut Grabner, Michael Grabner, and Horst Bischof. Real-time tracking via on-line boosting. In *Bmvc*, volume 1, page 6, 2006.

[29] Gültekin Gündüz and Tankut Acarman. A lightweight online multiple object vehicle tracking method. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 427–432. IEEE, 2018.

[30] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[31] Mehrtash Harandi, Soumava Roy Kumar, and Richard Nock. Siamese networks: A thing or two to know. 2017.

[32] Anfeng He, Chong Luo, Xinmei Tian, and Wenjun Zeng. A twofold siamese network for real-time object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4834–4843, 2018.

[33] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 1026–1034, 2015.

[34] Elad Hoffer and Nir Ailon. Deep metric learning using triplet network. In *International Workshop on Similarity-Based Pattern Recognition*, pages 84–92. Springer, 2015.

[35] Weiming Hu, Xi Li, Wenhan Luo, Xiaoqin Zhang, Stephen Maybank, and Zhongfei Zhang. Single and multiple object tracking using log-euclidean riemannian subspace and block-division appearance model. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(12):2420–2440, 2012.

[36] Braden Hurl, Krzysztof Czarnecki, and Steven Waslander. Precise synthetic image and lidar (presil) dataset for autonomous vehicle perception. In *2019 IEEE Intelligent Vehicles Symposium (IV) (In Press)*. IEEE, 2019.

[37] Mingxin Jiang, Zhenzhou Tang, and Liming Chen. Tracking multiple targets based on min-cost network flows with detection in rgb-d data. *International Journal of Computational Science and Engineering*, 15(3-4):330–339, 2017.

[38] Andrej Karpathy. Cs231n convolutional neural networks for visual recognition.

[39] Jonathan Kelly and Steven Waslander. Coursera: Lesson 1: The (linear) kalman filter - module 2: State estimation - linear and nonlinear kalman filters. https://www.coursera.org/learn/state-estimation-localization-self-driving-cars/lecture/7DFmY/lesson-1-the-linear-kalman-filter.

[40] Chanho Kim, Fuxin Li, Arridhana Ciptadi, and James M. Rehg. Multiple hypothesis tracking revisited. *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4696–4704, 2015.

[41] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. arxiv:1412.6980, 2014.

[42] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2, 2015.

[43] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[44] Jason Ku, Melissa Mozifian, Jungwook Lee, Ali Harakeh, and Steven L Waslander. Joint 3d proposal generation and object detection from view aggregation. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–8. IEEE, 2018.

[45] Harold W Kuhn. The hungarian method for the assignment problem. *Naval research logistics quarterly*, 2(1-2):83–97, 1955.

[46] Roger Labbe. Kalmanfilter - filterpy 1.4.4 documentation. https://filterpy.readthedocs.io/en/latest/kalman/KalmanFilter.html#id2.

[47] Laura Leal-Taixé, Cristian Canton-Ferrer, and Konrad Schindler. Learning by tracking: Siamese cnn for robust target association. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 33–40, 2016.

[48] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[49] X Rong Li and Yaakov Bar-Shalom. Stability evaluation and track life of the pdaf for tracking in clutter. *IEEE Transactions on Automatic Control*, 36(5):588–602, 1991.

[50] Xin Li, Kejun Wang, Wei Wang, and Yang Li. A multiple object tracking method using kalman filter. In *The 2010 IEEE international conference on information and automation*, pages 1862–1866. IEEE, 2010.

[51] Yuan Li, Chang Huang, and Ram Nevatia. Learning to associate: Hybridboosted multi-target tracker for crowded scene. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 2953–2960. IEEE, 2009.

[52] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[53] T.-Y. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *ICCV*, pages 2999–3007, 2017.

[54] Wenhan Luo, Junliang Xing, Anton Milan, Xiaoqin Zhang, Wei Liu, Xiaowei Zhao, and Tae-Kyun Kim. Multiple object tracking: A literature review. *arXiv preprint arXiv:1409.7618*, 2014.

[55] Mahendra Mallick and Barbara La Scala. Comparison of single-point and two-point difference track initiation algorithms using position measurements. *Acta Automatica Sinica*, 34(3):258–265, 2008.

[56] Anton Milan, Seyed Hamid Rezatofighi, Anthony R. Dick, Konrad Schindler, and Ian D. Reid. Online multi-target tracking using recurrent neural networks. *CoRR*, abs/1604.03635, 2016.

[57] R. Mobus and U. Kolbe. Multi-target multi-object tracking, sensor fusion of radar and infrared. In *IEEE Intelligent Vehicles Symposium, 2004*, pages 732–737, June 2004.

[58] Kevin P Murphy. *Machine learning: a probabilistic perspective*. MIT press, 2012.

[59] Sang-Il Oh and Hang-Bong Kang. Multiple objects fusion tracker using a matching network for adaptively represented instance pairs. *Sensors (Basel, Switzerland)*, 17, 04 2017.

[60] Kenji Okuma, Ali Taleghani, Nando De Freitas, James J Little, and David G Lowe. A boosted particle filter: Multitarget detection and tracking. In *European conference on computer vision*, pages 28–39. Springer, 2004.

[61] A. O. Pak, Javier Correa, Martin Adams, Daniel Clark, Emmanuel Delande, Jérémie Houssineau, and Jose Franco. Joint target detection and tracking filter for chilbolton advanced meteorological radar data processing. In *Advanced Maui Optical and Space Surveillance Technologies Conference*, 2016.

[62] Luis Perez and Jason Wang. The effectiveness of data augmentation in image classification using deep learning, 2017.

[63] Zhen Qin and Christian R Shelton. Improving multi-target tracking via social grouping. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1972–1978. IEEE, 2012.

[64] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.

[65] Donald Reid. An algorithm for tracking multiple targets. *IEEE transactions on Automatic Control*, 24(6):843–854, 1979.

[66] B. Ristic and M. L. Hernandez. Tracking systems. In *2008 IEEE Radar Conference*, pages 1–2, May 2008.

[67] Lorenzo Rosasco, Ernesto De Vito, Andrea Caponnetto, Michele Piana, and Alessandro Verri. Are loss functions all the same? *Neural Computation*, 16(5):1063–1076, 2004.

[68] David A Ross, Jongwoo Lim, Ruei-Sung Lin, and Ming-Hsuan Yang. Incremental learning for robust visual tracking. *International journal of computer vision*, 77(1-3):125–141, 2008.

[69] David E Rumelhart, Geoffrey E Hinton, Ronald J Williams, et al. Learning representations by back-propagating errors. *Cognitive modeling*, 5(3):1, 1988.

[70] H. Ryuhei, F. Aito, N. Keisuke, I. Tomoyuki, and H. Shuhei. Effective use of dilated convolutions for segmenting small object instances in remote sensing imagery. In *WACV*, pages 1442–1450, 2018.

[71] Sumit Saha and Sumit Saha. A comprehensive guide to convolutional neural networks-the eli5 way, Dec 2018. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[72] Samuel Scheidegger, Joachim Benjaminsson, Emil Rosenberg, Amrit Krishnan, and Karl Granström. Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering. In *2018 IEEE Intelligent Vehicles Symposium (IV)*, pages 433–440. IEEE, 2018.

[73] Sarthak Sharma, Junaid Ahmed Ansari, J. Krishna Murthy, and K. Madhava Krishna. Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking. *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3508–3515, 2018.

[74] Rishabh Shukla. L1 vs. l2 loss function rishabh shukla. http://rishy.github.io/ml/2015/07/28/l1-vs-l2-loss/, July 2015.

[75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[76] Bi Song, Ting-Yueh Jeng, Elliot Staudt, and Amit K. Roy-Chowdhury. A stochastic graph evolution framework for robust multi-target tracking. In *ECCV*, 2010.

[77] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[78] L. Svensson, D. Svensson, and P. Willett. Set jpda algorithm for tracking unordered sets of targets. In *2009 12th International Conference on Information Fusion*, pages 1187–1194, July 2009.

[79] Daniel Svozil, Vladimir Kvasnicka, and Jiri Pospichal. Introduction to multi-layer feed-forward neural networks. *Chemometrics and intelligent laboratory systems*, 39(1):43–62, 1997.

[80] Ran Tao, Efstratios Gavves, and Arnold W. M. Smeulders. Siamese instance search for tracking. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1420–1429, 2016.

[81] Wei Tian, Martin Lauer, and Long Chen. Online multi-object tracking using joint domain information in traffic scenarios. *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[82] Tsang. Review: Dilatednet - dilated convolution (semantic segmentation), Nov 2018. https://towardsdatascience.com/review-dilated-convolution-semantic-segmentation-9d5a5bd768f5.

[83] Rahul Rama Varior, Mrinal Haloi, and Gang Wang. Gated siamese convolutional neural network architecture for human re-identification. *ArXiv*, abs/1607.08378, 2016.

[84] Shaofei Wang and Charless C. Fowlkes. Learning optimal parameters for multi-target tracking with contextual interactions. *International Journal of Computer Vision*, 122:484–501, 2016.

[85] Bo Wu and Ram Nevatia. Detection and tracking of multiple, partially occluded humans by bayesian combination of edgelet based part detectors. *International Journal of Computer Vision*, 75(2):247–266, 2007.

[86] Yu Xiang, Alexandre Alahi, and Silvio Savarese. Learning to track: Online multi-object tracking by decision making. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[87] J. Xiao, H. Cheng, H. Sawhney, and F. Han. Vehicle detection and tracking in wide field-of-view aerial video. In *2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 679–684, June 2010.

[88] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *arXiv preprint arXiv:1511.07122*, 2015.

[89] Amir Roshan Zamir, Afshin Dehghan, and Mubarak Shah. Gmcp-tracker: Global multi-object tracking using generalized minimum clique graphs. In *European Conference on Computer Vision*, pages 343–356. Springer, 2012.

[90] L. Zhang and L. van der Maaten. Structure preserving object tracking. In *2013 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1838–1845, June 2013.

[91] Li Zhang, Yuan Li, and Ramakant Nevatia. Global data association for multi-object tracking using network flows. In *2008 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8. IEEE, 2008.

[92] Xizhou Zhu, Yuwen Xiong, Jifeng Dai, Lu Yuan, and Yichen Wei. Deep feature flow for video recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2349–2358, 2017.