Onboard ship detection and pose estimation with deep learning

Liu, Z.

Award date:
2018

Technische Universiteit
Eindhoven
University of Technology

Department of Mechanical Engineering
System and Control
Research Group Control Systems

Onboard Ship Detection and Pose Estimation with Deep Learning

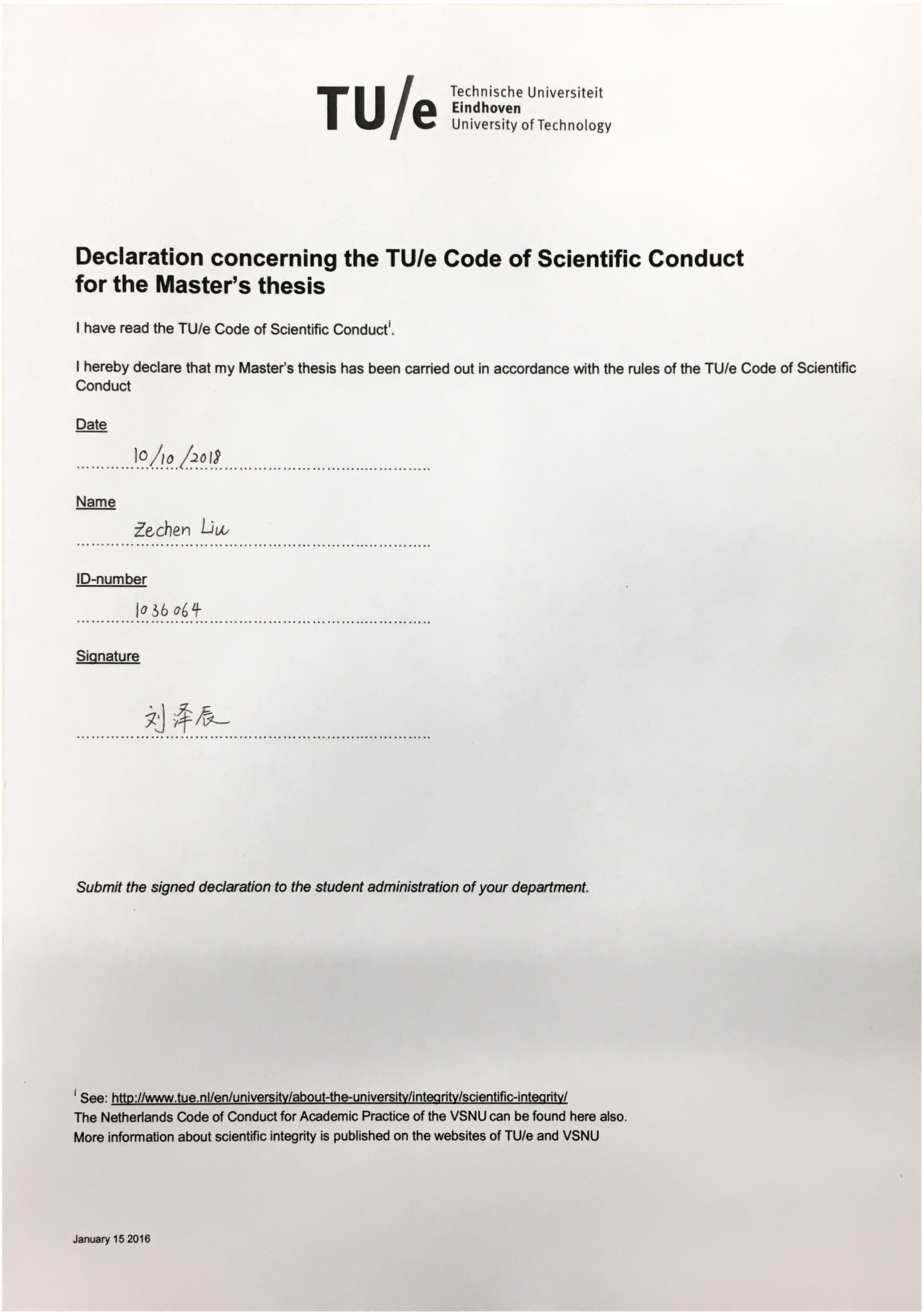
Master thesis

Zechen Liu

Supervisors:
dr.ir. R. Toth
ir. S. Rooijakkers

3rd version

Eindhoven, September 2018



TU/e Technische Universiteit **Eindhoven** University of Technology

Declaration concerning the TU/e Code of Scientific Conduct for the Master's thesis

I have read the TU/e Code of Scientific Conduct[i].

I hereby declare that my Master's thesis has been carried out in accordance with the rules of the TU/e Code of Scientific Conduct

Date

10/10/2018

Name

Zechen Liu

ID-number

1036064

Signature

刘泽辰

Submit the signed declaration to the student administration of your department.

[i] See: http://www.tue.nl/en/university/about-the-university/integrity/scientific-integrity/
The Netherlands Code of Conduct for Academic Practice of the VSNU can be found here also.
More information about scientific integrity is published on the websites of TU/e and VSNU

January 15 2016

Abstract

Perception is a crucial factor for intelligent robots as well as autonomous vehicles since it can extract a wide range of sensory information from the surrounding environment. Among all the perception methods, object detection is an essential component as it simultaneously predicts object's category and location. In this thesis, we propose a two-stage Convolutional Neural Network based detection method along with sensor fusion technique which can estimate 2D and 3D bounding box of ships separately. The method can predict useful information such as orientation and location of ships. We first modify and implement well-known 2D detection Faster R-CNN [8] model on different types of ships. Based on the 2D detection results, a state-of-art 3D bounding box estimation [16] approach is added to predict orientation, dimensions, and locations for different objects. Last, a Kalman Filter based sensor fusion method is implemented to refine 3D information by fusing Lidar and vision information together.

Various datasets are used to test the performance of our method. The 2D detection performance achieves 0.728 mean average precision on three specific types of ships from Damen shipyards. The 3D detection results in nearly 11° orientation estimation error on a toy ship. The sensor fusion method largely reduces the prediction error from a single camera. Based on the results, we discuss the limitations of our approaches and point out the possible future development directions.

Acknowledgement

I would like to express my sincere gratitude to my supervisor dr.ir.Roland Toth and Damen shipyards company for providing me such an interesting topic to work on. Dr.ir.Roland Toth guided me through this project and helped me a lot in academic research. I would also like to thank my local supervisor Siebe Rooijakkers for helping me with all the technical problems I encountered at Damen. Finally, I must express my deep and profound gratitude to my parents for their unlimited support, continuous encouragement and unconditional love throughout my life and especially through my two-year master study in the Netherlands.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

With the rapid development of science and technology, it is widely believed that automation will revolutionize the world and improve the living standard of human beings in many aspects. In maritime technology, many shipping companies hold the belief that autonomous shipping is the future of the ship industry since it can significantly lower the cost of shipping, increase onboard safety, and improve working conditions. Research has already been conducted on developing autonomous vessels. An overview of the research direction and autonomy levels definition can be found at [1, 2, 3]. Deep learning based computer vision to reach the aimed autonomy levels such as ship classification and ship detection, has also been developed recently [3, 4]. At Damen shipyards, many research projects on autonomous navigation and control vessels are being conducted and pointing towards the realization of ships with fully autonomous system in the next few years.

Autonomous vehicles are equipped with intelligent autonomous systems which collect useful information from the outside environment, process it and set target actions, and finally send commands to the actuators to execute. The core competencies of such an intelligent autonomous system on the software level can be broadly separated into three parts [5, 6]: perception, planning, and control. Perception refers to the ability of an intelligent system to receive information and extract useful knowledge from it. Planning is the procedure of making purposeful decisions to achieve higher order goals such as obstacle avoidance or path optimization. The control competency is the ability of the system to execute the planned actions. In this project, we mainly investigate and study the perception part for autonomous vessels.

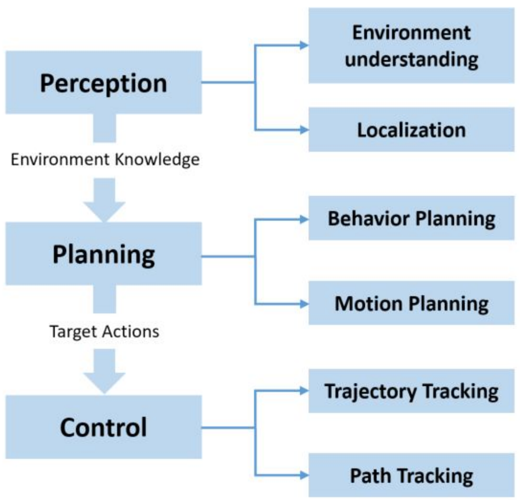


Figure 1.1: Overview of a typical intelligent autonomous system overview on software level [5].

Object detection is a key aspect of autonomous vehicle perception since it can perceive the surrounding environment with fitting bounding box on object instances and classifying of them into categories. There has been extensive study and research conducted on visual detection and many classical computer vision approaches exist [7, 8, 9]. Recently, with the development in deep learning, especially in the filed of Convolutional Neural Network (ConvNet), object detection has seen a significant progress [10, 11]. ConvNet based detection algorithm such as Faster R-CNN [12] and SSD [13] has achieved promising results on both detection accuracy and speed. These methods have been adopted in many practical applications. However, they only output 2D bounding box on objects which is insufficient for autonomous vehicles.

3D object detection is of particular importance for intelligent autonomous systems because it can output adequate information including pose and locations of different objects. Many research work in recent years has focused on the development of accurate and robust 3D bounding box estimation [14, 15, 16]. All these methods achieve decent and accurate detection results, but they all require difficult obtainable training information such as Lidar point cloud or computational preprocessing work.

In order to achieve a high accuracy of detection and pose estimation for autonomous vehicles, additional sensors can be equipped to achieve multiple forms of perception. Sensors such as LiDAR, radar and Inertial Measurement Unit (IMU) are always used onboard to provide helpful information besides camera-based vision [17]. Therefore, combining information from multiple sensors together is necessary and important for improving the perception capability of the system.

1.1 Problem Statement

In order to enhance ship navigation and development of autonomous ships, the goal of this project is to develop a ConvNet based 3D object detection method with respect to ship detection, implement and validate it on available datasets. Furthermore, we investigate if a monocular camera-based detection can be used to obtain accurate 3D information. Additionally, we aim to investigate how useful information from various sensors of the ship can be fused together with the 3D object detection to further increase the performance of localization and orientation estimation.

1.2 Proposed Solutions

In our approach, we develop a two-stage detection algorithm which predicts 2D bounding box and 3D bounding box separately based on the state-of-art 2D and 3D object detection methods [18, 19, 12, 20]. Each stage of the detection algorithm can be trained independently and easily improved by attaching more advanced ConvNet on it to extract features. The predicted outputs from the detection method are fused together with Lidar information by applying a linear time-invariant Kalman Filter. As a result, we achieve 0.728 mean average precision (mAP) 2D ship detection result on three specific types of ships from Damen shipyards. The 3D ship detection method results in around $11°$ mean error on a toy ship dataset. The Kalman Filter based sensor fusion method further improves the estimation accuracy of 3D information.

1.3 Overview

In the second chapter, a brief introduction into the theory of ConvNet is provided along with related work in 2D and 3D object detection. In addition, we explain and motivate the algorithm choice in detail. The next chapter describes the approach that is proposed and developed in our work together with an overview of the resulting ConvNet structure. In chapter 4, the description of the datasets and data representations used for training are discussed. To demonstrate the properties of the developed method, the experiment settings, evaluation metrics and results are reported at the end of this chapter. Finally, we summarize our work and give the possible directions for further research in the last chapter.

Chapter 2

Preliminaries

In this chapter, the fundamental concepts of deep learning are introduced together with a brief introduction to Multi-layer Perception (MLP, or Artificial Neural Network), Convolutional Neural Network (ConvNet), and their usage in object detection field.

2.1 Multi-layer Perceptions

In order to describe Neural Networks, we start with introducing the basic computation of a single neuron unit in MLP. MLP was originally introduced by neurophysiologists inspired by biology, see Hubel and Wiesel's early work [21]. They were invented to mimic the visual cortex of a cat. Based on the biological neuron model, the mathematical model of a neuron receives input and sums these values up with weights and bias. If the sum exceeds a specific threshold function, the signal will be activated and sent to the unit in the next layer.

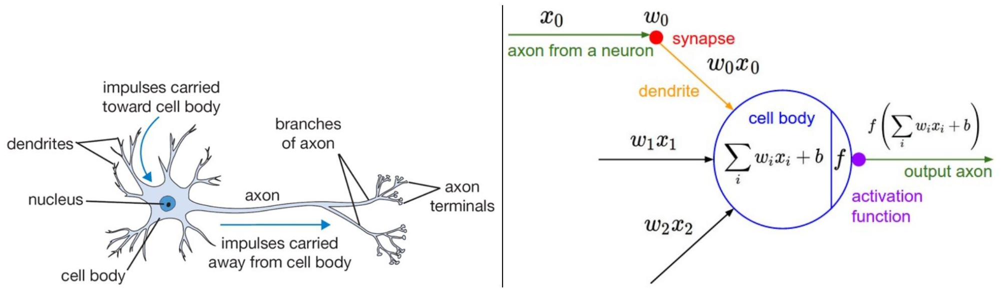


Figure 2.1: Left: Cartoon model of a biological neuron. Right: The mathematical model [22].

As shown in Figure 2.1, a neuron accepts $x = [x_0\, x_1\, \cdots\, x_n]^\top$ as inputs and outputs $y = f(\sum_i w_ix_i + b)$, where weights w_i and bias b are learning parameters in this neuron model. f is a nonlinear activation function with one of the mathematical descriptions given in Table 2.1.

Name	Equation
Sigmoid	$f(z) = \frac{1}{1+e^z}$
Tanh	$f(z) = \frac{e^z-e^{-z}}{e^z+e^{-z}}$
ReLu[23]	$f(z) = \max(0, z)$
LeakyReLu[24]	$f(z) = \begin{cases} 0.1z, z < 0 \\ z, z \geq 0 \end{cases}$

Table 2.1: Choices of activation functions.

The MLP structure is modeled as collections of neurons that are connected in an acyclic graph. Neurons between two adjacent layers are pairwise connected, but neurons in a single layer share no connections. The forward pass of one layer corresponds to a matrix multiplication followed by a bias offset and an activation function. This can be expressed by the following equation:

$$y = f_l(f_{l-1}(f_{l-2}(...(x))). \tag{2.1}$$

Normally an MLP system contains one input layer, several hidden layers and one output layer. A typical MLP system is shown in Figure 2.2.

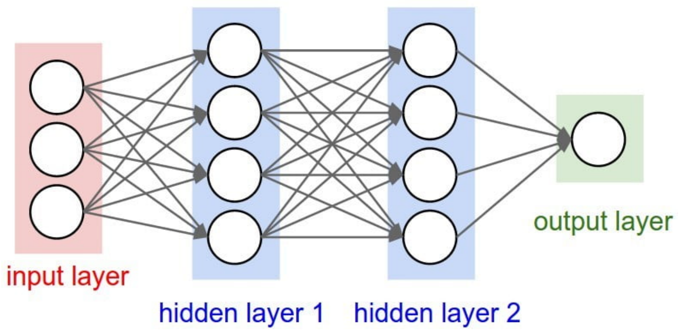


Figure 2.2: A three-layer neural network with three inputs, two hidden layers of 4 neurons each and one output layer [22].

Notice that a loss function is usually applied to the final layer of the Neural Network instead of the activation function. This function is used to represent the performance of MLP. After constructing the MLP structure, it needs to be trained in order to function, which means appropriate tuning of the weights w and bias b in the network. A useful method of training is using a gradient descent optimization algorithm such as SGD [25] or Adam [26] to propagate error from output units back to the weights and bias at each layer. This procedure is called back-propagation.

When a network using activation function σ is trained , the neuron unit j in layer l can be described by the weight sum a_j^l and activation output z_j^l as defined in Equation 2.2 and 2.4.

$$a_j^l = \sum_i w_{ij}^l z_i^{l-1}, \tag{2.2}$$

where w_{ij}^l is the weight between neuron i at layer $l-1$ and neuron j at layer l. For simplicity, the bias b is ignored here.

$$z_j^l = \sigma(a_j^l), \tag{2.3}$$
$$\mathcal{L} = L(a^L). \tag{2.4}$$

$\mathcal{L}$ is the output of the loss function L. a^L is a vector that contains all output neuron units $[a_0^L \, a_1^L \, \cdots \, a_n^L]^\top$. The partial derivatives of the loss function with respect to the weights can be computed as follows:

$$\begin{aligned}\frac{\partial \mathcal{L}}{\partial w_{ij}^l} &= \frac{\partial \mathcal{L}}{\partial a_j^l} \frac{\partial a_j^l}{\partial w_{ij}^l}, \\ &= \delta_j^l z_i^{l-1}.\end{aligned} \tag{2.5}$$

The term δ_j^l is back propagated during training. By using the chain rule of differentiation, δ_j^l

satisfies the following recursive equation:

$$\begin{aligned}
\delta_j^l &= \frac{\partial \mathcal{L}}{\partial a_j^l}, \\
&= \sum_k \frac{\partial \mathcal{L}}{\partial a_k^{l+1}} \frac{\partial a_k^{l+1}}{\partial a_j^l}, \\
&= \sum_k \delta_k^{l+1} \frac{\partial}{\partial a_j^l} \sum_j w_{jk}^{l+1} z_j^l, \\
&= \sum_k \delta_k^{l+1} w_{jk}^{l+1} \sigma'(a_j^l).
\end{aligned} \tag{2.6}$$

The δ term in a certain hidden layer of Neural Networks can be derived by propagating δ from a higher layer. Since the output value of the final layer is known, the error for units in the initial layers can be obtained recursively. After computing the gradient, the updated weights can be calculated with the learning rate γ.

$$w_{jk}^l = w_{jk}^l - \gamma \frac{\partial \mathcal{L}}{\partial w_{jk}^l}. \tag{2.7}$$

The parameters in MLP can be iteratively updated by using Equation 2.7 until the model can reach a certain target performance.

2.2 Convolutional Neural Networks

ConvNet is mainly used in visual data processing such as images and videos. Their application has achieved significant results on tasks such as image recognition, object detection, semantic segmentation, video understanding, etc. Almost all the state-of-art methods in these fields are ConvNet based algorithms. The first ConvNet was introduced by Yann LeCun's LeNet [27] which was used to recognize various digits on envelopes. The structure of LeNet is shown in Figure 2.3. Different from MLP, the neurons in a layer will only be connected to a small region of the layer before it, which results in local connectivity and weights sharing properties [28]. A simple ConvNet contains three main types to build its structure: Convolutional Layer, Pooling Layer, and Fully-Connected Layer. The structure and effect of these layers will be discussed in this section.

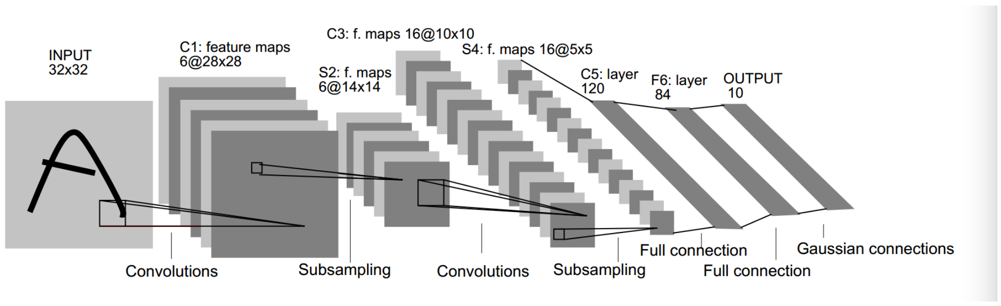


Figure 2.3: The Architecture of Lenet [27].

2.2.1 Input Layer

Unlike normal MLP, which receives a single vector as input and forward propagates it through a series of hidden layers, ConvNet receives original images as input and generates data in terms of 3-dimensional shapes. An RGB image can be described as a structure of $W \times H \times C$, where W

and H are the number of pixels in width and height of the image, and C denotes the number of channels. If the image is reshaped into a vector and an MLP architecture mentioned above is built for classification, the number of trainable parameters will be extremely large. Hence, along with computational problems, over-fitting is likely to happen. Furthermore, such a naive MLP model would eliminate the 3D structure information of the input images.

Some data preprocessing methods have been implemented on the input layer to improve the performance of ConvNet. The most commonly used approaches are mean subtraction and normalization [29, 30].

2.2.2 Convolutional Layer

Convolutional layer contains a set of learnable filters. Each filter is small in width and height, but extends through the full depth of the input volume. For example, in the typical ConvNet LeNet [27], a first convolutional layer has size $5 \times 5 \times 3$ (i.e. 5 pixels in width and height, 3 for RGB image channels). In the forward pass, the filter slides across the width and height of the input and compute dot product between the filter itself and the input at different positions. As a result, each filter will produce a 2-dimensional activation map as output. Notice that each output neuron is only connected to a local region in the previous layer. The resulting convolutional layer has the following properties:

Local connectivity: Similar to input images, the hidden layers in ConvNet are also in three dimensions: width, height, and channels. Furthermore, each neuron unit inside a layer only connects to a certain region of the previous layer which is called the receptive field of the neuron. Therefore, each neuron only has a connection with a sub-region in the previous layer instead of the whole channel regions.

Spatial determination: The size of the output volume is decided by three hyper-parameters: depth (K), stride (S), and zero-padding (P). The number of filters is the depth of the output volume. Each filter learns to discover different information from the input. In LeNet [27], the depth of the first batch of layers is 6. Each extracts information from the input images which result in 6 output volumes. On the other hand, the stride value determines the width and height of the output. When the stride is 1, the filter will only move 1 pixel at a time. A large stride will results in smaller volumes spatially. Zero-padding is always used to extend the spatial size of the input volume so that the output size can be controlled. The property of zero-padding enables the size of input volume can be changed easily without adding other noisy information to it.

Weights sharing: Additionally, the connections between a neuron and its corresponding receptive field refer to one filter. Each filter slides across the entire input volume, which means that the neurons in the same depth slice share the same filter. In other words, each depth slice reveals the responses of one filter applied to the previous layer. Normally a depth slice is referred as a feature map, and thus a layer consists of several feature maps.

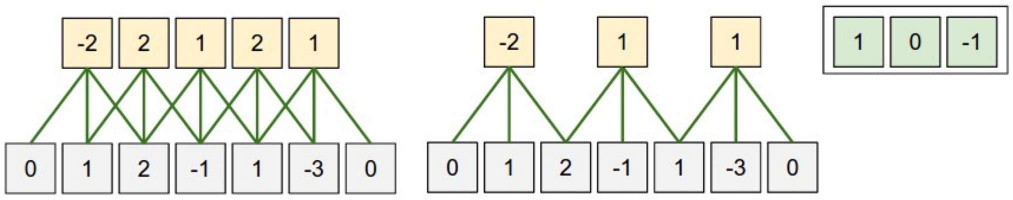


Figure 2.4: Illustration of the convolutional layer properties in one spatial dimension. The filter is displayed in green color with size $F = 3$. The input volume is $W = 5$, zero-padding is set to 1. Left: the filter slides with $S = 1$. Right: the filter slides with $S = 2$ [22].

To conclude, if multiple filters are given, the convolution operation will output a stack of feature maps to form a new convolution layer in 3D. For example, one convolutional layer has size $W_1 \times H_1 \times C_1$. K filters with size $F \times F \times C_1$ slides across this layer and yields an output layer with size $W_2 \times H_2 \times C_2$. These three values can be determined by the following equations:

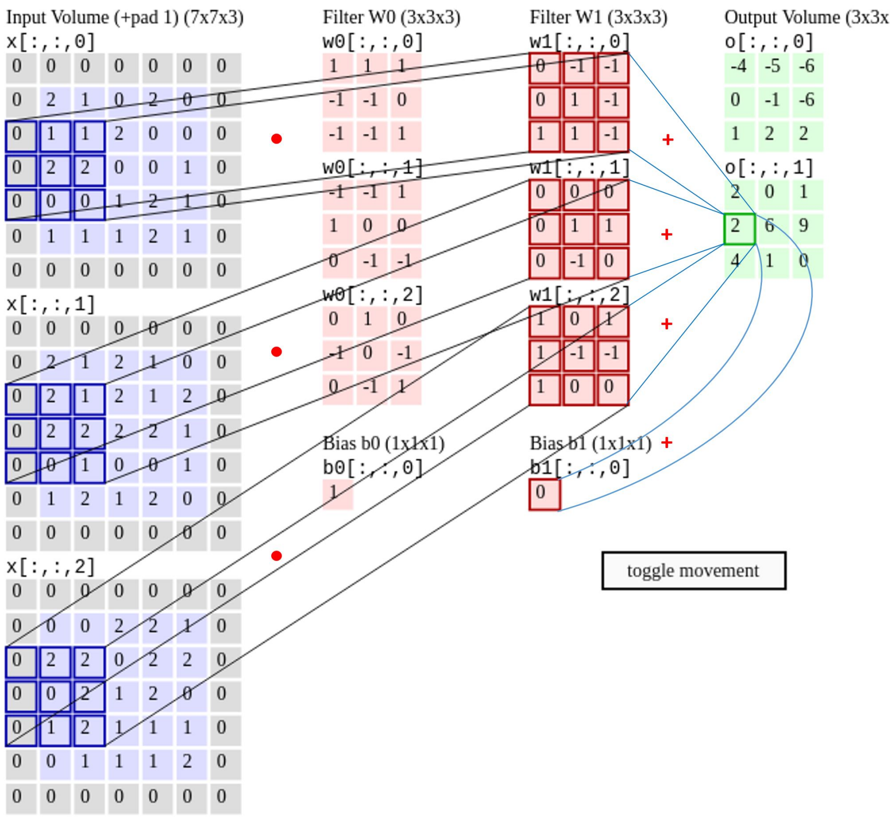


Figure 2.5: Explanation of how convolutional layer works on images. input layer: $5 \times 5 \times 3$ with 1 zero padding, two filters: $3 \times 3 \times 3$, stride: 2, output layer: $3 \times 3 \times 2$. Each number in output layer is computed by an element-wise multiplication between the filter and a certain region in the input layer. The output values are summed up together with the bias [22].

$$\begin{aligned} W_2 &= (W_1 - F + 2P)/S + 1, \\ H_2 &= (H_1 - F + 2P)/S + 1, \\ C_2 &= K. \end{aligned} \tag{2.8}$$

with weight sharing, this operation totally introduces $(F \times F \times C_1) \times K$ weights and K biases. A detailed illustration of how the convolutional layer works on images is shown in Figure 2.5.

2.2.3 Pooling Layer

Pooling layer (or downsampling) is inserted periodically between a sequence of convolution layers. It is used to reduce the spatial size of feature maps hence reduce the number of parameters and computation in training ConvNet and also control overfitting. The pooling layer operates on every channel of feature maps independently. Common pooling methods include max pooling, average pooling and l_2 norm pooling. The most common form of the pooling layer includes filters with 2×2 size with a stride of 2. In ConvNet, a pooling layer accepts an input volume of size $W_1 \times H_1 \times C_1$. The spatial extent of a pooling layer is F and the stride is S. The output feature maps has size $W_2 \times H_2 \times C_2$. These three parameters can be determined by:

$$\begin{aligned} W_2 &= (W_1 - F)/S + 1, \\ H_2 &= (H_1 - F)/S + 1, \\ C_2 &= C_1. \end{aligned} \tag{2.9}$$

The most often used pooling operation now is max pooling due to the simplicity and high performance. Figure 2.6 shows how max pooling works in ConvNet. However, some researchers [31] suggest use convolutional layer with a large stride to reduce spatial size instead of adding a pooling layer.

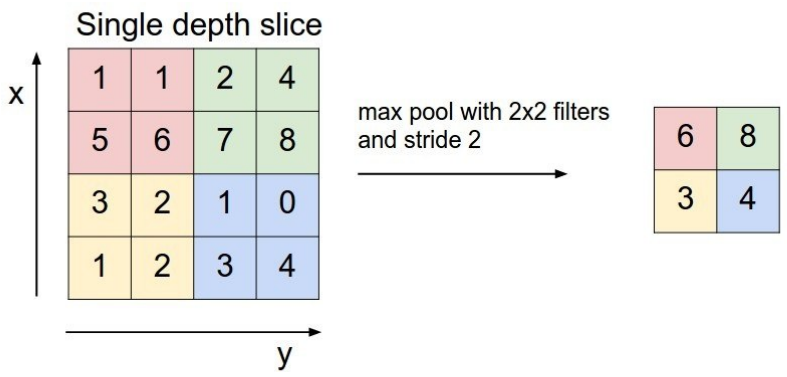


Figure 2.6: Max pooling layer downsamples each feature map spatially, regardless of the depth. The displayed max pooling layer is with $F = 2$ and $S = 2$ [22].

2.2.4 Activation Layer

An activation function (in Table 2.1) is normally added after convolutional layer to increase non-linearity representation of ConvNet. It is element-wise applied to each neuron. The input volume and output feature maps share the same spatial structure. Notice that the mostly used activation function is ReLu [23] in ConvNet because of its high-efficiency and large avoidance of gradient vanishing problems [32, 33]. The ReLu activation layer has the following operation:

$$f(z) = \max(0, z), \tag{2.10}$$

where z represents each neuron. An overview of activation function and the advantage of ReLu can be found in [11]. Furthermore, some researchers have claimed that more accurate results can be achieved by using LeakyReLu [24] with a small slope at negative values. However, the results of Leakyrelu are not always consistent.

2.2.5 Fully-connected Layer

The fully connected layer is usually inserted after the last pooling layer in ConvNet to reduce the number of feature maps and create a vector-like representation. Each neuron in this layer is fully-connected to all neurons in the last pooling layer like in MLP. It provides a form of dense connectivity and loses the structural layout of the input image. In classification tasks, this layer computes the final class scores where each class corresponds to a label in the training dataset. Figure 2.7 shows the operation of fully-connected layer.

2.2.6 Other Layers

With the rapid development of the deep learning field, many other operations have been developed to accelerate the training procedure, prevent ConvNet from overfitting, or avoid gradient vanishing

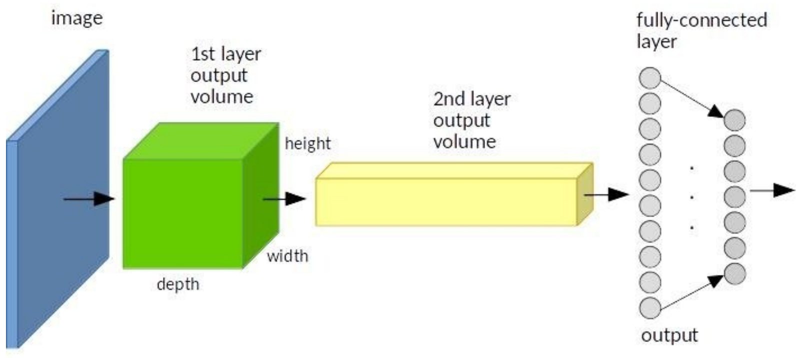


Figure 2.7: The yellow volume in a 3D shape is reshaped in to a vector-like representation. Each neuron is pairwise connected to neuron in the next layer as in MLP [34].

problems [32, 33]. Overfitting refers to that ConvNet reaches a high performance on the training set, but outputs inaccurate result on the test set. Gradient vanishing corresponds to the problem when the gradient function with respect to parameters in the first few layers becomes close to zero during back-propagation. Among all the operations, the most commonly used two operations to avoid these problems are Batch Normalization [35] and Dropout [36].

Batch Normalization: During training, the input images are preprocessed with mean subtraction and normalization as described in Section 2.2.1. ConvNet gains high performance and fast training speed with this operation. As for hidden layers, they tend to work better with received data consisting of features with zero mean and unit variance. However, the data in hidden layers is different and varies at each step due to the parameter updating by back-propagation. Batch normalization solves this problem by generating proper initialization for data in hidden layers. It explicitly forces the output of a convolutional layer to have a unit Gaussian distribution before the activation operation. This enforcement is inserted between convolutional layer (or fully-connected layer) and the connecting activation layer. Consider a mini-batch of feature maps $\mathcal{B} = \{x_1, \cdots, x_m\}$ after the convolutional layer operation, the mean $\mu_{\mathcal{B}}$,variance $\sigma_{\mathcal{B}}$ and normalized value $\hat{x}_i$ are computed by the following equation:

$$\begin{aligned}\mu_{\mathcal{B}} &= \frac{1}{m}\sum_{i=1}^{m} x_i,\\ \sigma_{\mathcal{B}}^2 &= \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_{\mathcal{B}})^2,\\ \hat{x}_i &= \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}},\end{aligned} \tag{2.11}$$

where ϵ is a small positive number. Batch normalization introduces another linear transformation y_i with two other learnable hyper-parameter scale γ and shift β. The number in $\hat{x}_i$ meets Gaussian distribution. Without this operation, most values are close to zero which could not be activated through nonlinear activation layer. In real application, the batch size m is set to 256 or 512 [29, 30]. A large or small batch size can result in performance degradation problem [37, 38]. The equation is:

$$y_i = \gamma\hat{x}_i + \beta. \tag{2.12}$$

The two factors γ and β can be learned and updated by ConvNet during training. Batch normalization makes ConvNet significantly more robust to bad initialization. Additionally, it can be interpreted as preprocessing before every layer of networks.

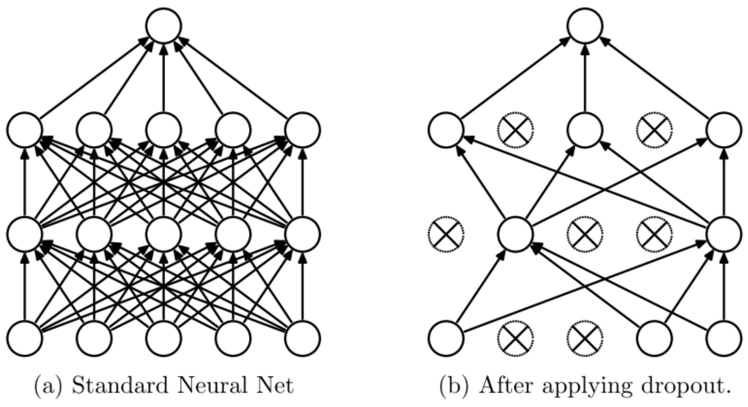


Figure 2.8: Image illustration of dropout. Left: A normal neural networks with 2 hidden layers. Right: Dropout neural networks with crossed units dropped [36].

Dropout: This method is an effective, simple, and recently introduced a technique to prevent overfitting. Figure 2.8 shows the idea of dropout. During training, dropout is implemented by only keeping one neuron unit active with probability p. It can be treated as a sampled ConvNet based on the original trained model and only parameters of the sampled ConvNet are updated based on the input data. Consider an output neuron z during training, the expected output of this neuron with dropout is:

$$\hat{z} = pz + (1-p)0, \tag{2.13}$$

where p is the probability between 0 and 1. In the test procedure, this neuron z is always active. However, the probability p must be multiplied on neuron z to keep the same expected output as in training process. Since the test time is critical, inverted dropout is more preferable to implement at training process. It performs scaling in training time and does not function at the testing procedure. The inverted dropout is illustrated as:

$$\begin{aligned}\hat{z} &= pz + (1-p)0,\\ \hat{z} &= \frac{z}{p}.\end{aligned} \tag{2.14}$$

2.3 Loss Function

ConvNet based object detection method is a supervised learning approach. The training data consists of images with label information characterizing the ground truth. During training, the loss function is used to measure the quality between predicted scores on training dataset and the ground truth labels. The target of training ConvNet is to minimize the loss function by updating weights and bias. The loss is computed by forward pass and the gradient of the network parameters is computed by back-propagation. The total loss is computed as an average over each training individual example:

$$\mathcal{L} = \frac{1}{N}\sum_i \mathcal{L}_i, \tag{2.15}$$

where N is the number of training data. Classification is to predict a discrete class label. Normally it is assumed that one example in the dataset only has one single correct label. The most commonly used loss function for classification is Softmax classifier with cross-entropy loss. Consider a training dataset that consists of n different labels. The output neuron units in last fully-connected layer is in a vector like form $[a_0^L \, a_1^L \, \cdots \, a_n^L]^\top$. The Softmax function is first used to transform these scores into a normalized vector:

$$p_i = \frac{e^{a_i^L}}{\sum_{j=1}^{n} e^{a_j^L}}. \tag{2.16}$$

The cross-entropy loss for one training example has the following form:

$$\mathcal{L}_i = -\log(p_i). \tag{2.17}$$

Regression is the objective of predicting continuous quantities such as the location of objects in images. The L_2 loss or L_1 loss is always used to compute the loss between the predicted quantity and the true label value. The L_2 loss has the following form:

$$\mathcal{L}_i = \|a_i^L - y_i\|_2^2. \tag{2.18}$$

The L_1 loss has the following equation:

$$\mathcal{L}_i = \|a_i^L - y_i\|, \tag{2.19}$$

where y_i is the true label value for i-th example. Note that L_2 loss is more difficult to optimize and less robust to outliers because it can introduce huge gradients. Also, regularization [39] is always used to prevent ConvNet from overfitting.

2.4 Examples of ConvNet Structure

ConvNet saw heavy use in the 1990s, but then faded away because of the increasing problem complexity and the limitation of computation capability. In 2012, AlexNet [11] brought ConvNet back with a huge improvement on image classification accuracy on ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [10] and high parallel GPU implementation. Since then, every year the winner of ImageNet has been a ConvNet. The most successful structures are VGG-16 [29], GoogLeNet [40], ResNet [30], DenseNet [41], etc. The most commonly used two architecture are VGG-16 and ResNet. Both ConvNets are used in scientific research and industry development. ResNet-50 takes less time to train and gain more accurate result than VGG-16 [42]. However, its shortcut connection breaks the successive ConvNet structure which makes it difficult for transfer learning.

2.4.1 VGG-16

VGG-16 was the runner-up ConvNet in ILSVRC 2014. Its main contribution was showing that the depth of ConvNet is a critical component for better performance. The filter size in the convolution layer is only 3×3 and the max pooling layer has size 2×2. The whole structure of VGG-16 is shown in Figure 2.9.

As can be seen from the figure, the number of filters is doubled when the spatial size of feature maps reduce half. Three fully-connected layers are stacked after the convolution layers with dropout regularization for the first two fully-connected layers. VGG-16 achieved high performance on image recognition problems with only 7.3% error. However, one downside of this ConvNet is it contains nearly 138 million parameters to train. Most of these parameters are between the last convolution layer and the first fully-connected layer. The last convolution net has 512 feature maps with a dimension of 7×7 and each neuron unit is fully-connected with the first fully connected layer with the width of 4096. This will introduce $512 \times 7 \times 7 \times 4096 = 103$ million parameters. This transition makes about 74% of the parameters of the network. Researchers found these fully-connected layers can be replaced by average pooling [44] without any significant performance downgrade while largely reducing the number of parameters.

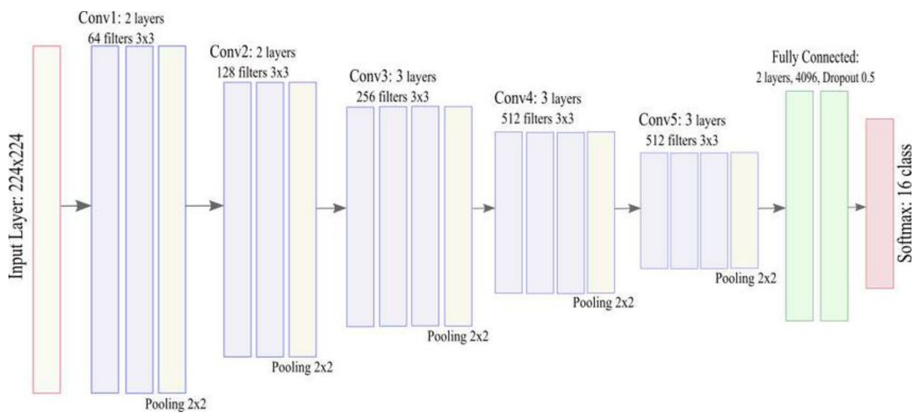


Figure 2.9: Architecture of VGG-16 [43].

2.4.2 ResNet

ResNet, developed by Microsoft Asia, was the winner of ILSVRC 2015. It introduced residual block (or shortcut connection) to improve classification accuracy on image recognition. The error performance of the winner network ResNet-152 is only 3.6%, which is even better than the performance of human beings. ResNet is currently the state-of-art ConvNet models according to its performance.

Since batch normalization can successfully solve gradient vanishing and explosion problem, it is logical that the ConvNet performance can be improved if more layers are stacked to it. However, based on experiments [45, 46], deeper networks face heavy degradation problem which means the accuracy of ConvNet degrades rapidly if the depth increases. For example, if we train the well pre-trained VGG-16 model with more layers added performing identity mapping, the result of the new deeper ConvNet will be worse than the original VGG-16 model. A residual block is introduced to address this problem. The structure is shown in Figure 2.10.

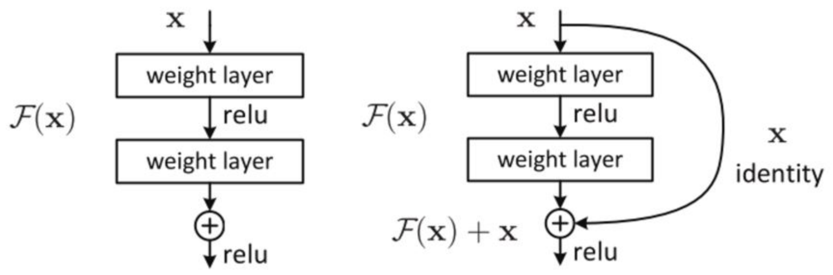

x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
relu
x
weight layer
$\mathcal{F}(\mathbf{x})$
relu
weight layer
x
identity
$\mathcal{F}(\mathbf{x}) + \mathbf{x}$
relu

Figure 2.10: Left: Two-layer standard network block. Right: Two layers residual network block 2.10.

The output of these two-layer networks is denoted as $\mathcal{H}(\mathbf{x})$. The original mapping of the left figure is recast into $\mathcal{H}(\mathbf{x}) = \mathcal{F}(\mathbf{x}) + \mathbf{x}$ as shown in the right figure. With this short cut property, the depth of ConvNet can be significantly increased. The central idea of ResNet is to learn the additive residual function $\mathcal{F}$ with respect to the identity mapping of $\mathbf{x}$. A more detailed explanation was given in [47]. Consider a ResNet with loss function $\mathcal{L}$, the back propagation to $\mathbf{x}$ is:

$$\begin{aligned} \frac{\partial \mathcal{L}}{\partial \mathbf{x}} &= \frac{\partial \mathcal{L}}{\partial \mathcal{H}} \frac{\partial \mathcal{H}}{\partial \mathbf{x}}, \\ &= \frac{\partial \mathcal{L}}{\partial \mathcal{H}}(1 + \mathcal{F}'(x)). \end{aligned} \tag{2.20}$$

This indicates that the gradient $\frac{\partial \mathcal{L}}{\partial \mathbf{x}}$ can be decomposed into two parts. The term $\frac{\partial \mathcal{L}}{\partial \mathcal{H}}$ can be propagated without concerning any other weight layers and scale. Another term $\frac{\partial \mathcal{L}}{\partial \mathcal{H}}\mathcal{F}'(x)$ propagates through the weight layers.

2.5 Deep Learning Framework

Recently many deep learning frameworks from companies or universities have been open sourced for the public. Most famous among these are Caffe [48], Pytorch [49], Tensorflow [50], MxNet [51], PaddlePaddle [52]. All these frameworks support GPU training with parallel computing and can be easily adapted for different tasks such as visual recognition and natural language processing.

In this project, all the work was carried out on Tensorflow and Keras [53]. TensorFlow is an open source software library for high performance numerical computation. It was originally developed by researchers and engineers from the Google Brain team within Googles AI organization. Furthermore, it comes with strong support for machine learning and deep learning and the flexible numerical computation core is used across many other scientific domains. In addition, Tensorflow orders 'TensorFlow Model Zoo' where people could get shared models and weights from Google research department, thus allowing researchers to build on top of others work. Keras is the official high-level application programming interface (API) of Tensorflow. It's growing with large adoption in the research community with its focus on user experience.

2.6 ConvNet Based Object Detection

While image classification focuses on recognizing candidate objects in images, object detection deals with detecting instances of semantic objects belonging to a certain class by fitting a bounding box on it. Object detection is a more challenging task since it also needs to identify the location of a certain instance along with its size. Furthermore, object locations in the image plane are insufficient for advanced tasks such as autonomous driving or robot navigation. In reality, 3D object detection is important since it can output useful 3D information for decision making and interaction. In this section, the recent advances ConvNet based approaches on object detection will be discussed. The difference between image recognition, 2D object detection, and 3D object detection are shown in Figure 2.11.

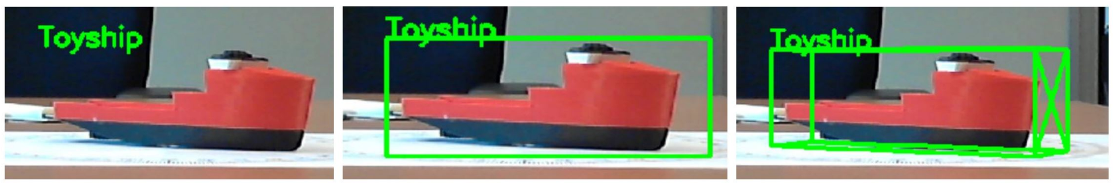


Figure 2.11: Left: Image classification. Middle: 2D object detection. Right: 3D object detection.

2.6.1 2D Object Detection

As described before, the most difficult problem by 2D object detection is finding object locations. A typical detection pipeline generally starts from searching for possible regions and extracting features region by region and then use classifiers to determine the object class for each region.

Many region proposal methods have been developed to address this task such as Selective Search [54], Edge boxes [55], and Region Proposal Networks (RPN) [12].

Region based Convolutional Neural Networks (R-CNN) [18] applies Selective Search to find regional object proposals. Each proposed region is then cropped from the original image and resized into 227×227. It is then fed into a 4096-dimensional feature vector based Neural Network feature extractor and then based on the extracted features, it is classified by a Support Vector Machine (SVM). R-CNN achieves accurate detection results on dataset such as PASCAL VOC [56]. However, it costs too much time to train because the Selective Search method is not efficient and every proposed region needs to go through the ConvNet to perform object classification and bounding box regression.

Fast R-CNN [19] speed up the computation of R-CNN largely. It only uses one single ConvNet on the entire input image and finds proposed regions on feature maps instead of on the original images. Also, the SVM on the region proposals are replaced by fully connected layers to predict class probability and bounding box coordinates simultaneously. Furthermore, a new type of layer called region of interest (RoI) pooling is introduced to connect feature maps and classifiers.

Instead of relying on an external object proposal method, Faster R-CNN [12] uses RPN which slides over the last convolutional feature maps to generate candidate anchors as bounding boxes in different scales and ration aspects. These proposals are then fed into Fast R-CNN to classify objects. Faster R-CNN is around 10 times faster than Fast R-CNN and can achieve about 10fps on a modern GPU.

In RetinaNet [57], a new region proposal method called Feature Pyramid Network (FPN) [58] is used as the backbone to generate a rich, multi-scale convolutional feature pyramid based on feature maps from different depth. Furthermore, it proposes a novel cross entropy loss to address the foreground-background class imbalance encountered during training FPN. It achieves better accuracy and speed than Faster R-CNN.

A novel approach called Mask R-CNN [59] can effectively detect objects in the image while generating a high quality segmentation mask for each detected object. It extends Faster R-CNN by adding another branch for predicting an object mask in parallel with the existing branch for bounding box recognition. In addition, Mask R-CNN can achieve advanced level tasks such as key point detection and human pose estimation. It can achieve around 5fps on a modern GPU.

Some other detection algorithms are proposal-free such as You Only Look Once (YOLO) [60] and Single Shot Detector (SSD) [13]. Rather than generating candidate anchors on last convolutional feature maps, they divide images into several grid cells. Within each of the grid cell, a set of base bounding boxes are generated. These proposal-free methods run much faster than region proposal based detection algorithm. However, their detection accuracy is around 10 percent lower than Faster R-CNN and RetinaNet.

2.6.2 3D Object Detection

Different from 2D object detection which only needs to estimate 4 parameters (x, y, w, h) for an object, 3D object detection needs to find 9 parameters $(x, y, z, \phi, \psi, \theta, w, h, l)$ to determine the location of one object, where x, y, z describe coordinates of the object in 3D space, ϕ, ψ, θ mean rotation of the object in terms of its body frame which are roll, pitch and yaw, w, h, l describe width, height and length of the object. Many research work has focused on 3D object detection these years.

3DOP [14] introduced a sophisticated detection framework. It uses the stereo information to generate 3D point cloud and then it is used to find the ground plane and score 3D box proposals. An energy function which encodes object size priors, ground plane and depth informed features are designed. The best-generated proposals are selected by minimizing the energy function.

Mono3D [61] is a modification of 3DOP method. It was made to exclude the need for stereo images. Although no point cloud is generated, the knowledge of the position for the ground plane is assumed. Also, fully convolutional networks (FCNs) from external sources are used to propose bounding boxes from object and class segmentation. Its performance is slightly worse than in the stereo case and the pipeline is even more complicated.

An interesting work called SubCNN [15] makes use of novel voxel-pattern-based representation, which represents cars of different viewpoints, occlusions, and truncations. Groups of such patterns represent subcategories, which are detected on the output of a SubCNN network. Interestingly, since the voxel patterns define spatial models, 3D segmentation and bounding box is obtainable from the prototype database. However, the net is still just a classifier, requiring a separating region-proposal stage.

Deep3DBox [20] introduces another approach. Based on 2D bounding box detection, they use ConvNet to regress the local orientation and object dimensions. After that, the estimated parameters are combined with geometric constraints to project a final 3D bounding box on the detected objects. Unlike previously described methods, during training only information provided in KITTI [62] annotation labels is used. The training and estimation procedure just based on monocular images. This is of particular interesting since in reality additional information such as stereo images or point cloud is not easy to obtain.

A novel approach is given by DeepMANTA [16], which can predict part localization, visibility characterization, and 3D dimension estimation simultaneously. The DeepMANTA network is able to localize vehicle parts even if these parts are not visible. In the inference, the networks outputs are used by a real-time robust pose estimation algorithm for fine orientation estimation and 3D vehicle localization. However, they use an additional dataset including 3D shape and template of different types of vehicles.

2.7 Comparision of Detection Algorithm Properties

A survey paper published by Google Research [63] investigated the trade-off between speed and accuracy for 2D object detection. Two typical detectors Faster R-CNN [12] and SSD [13] are compared and discussed. The result is shown in Figure 2.12.

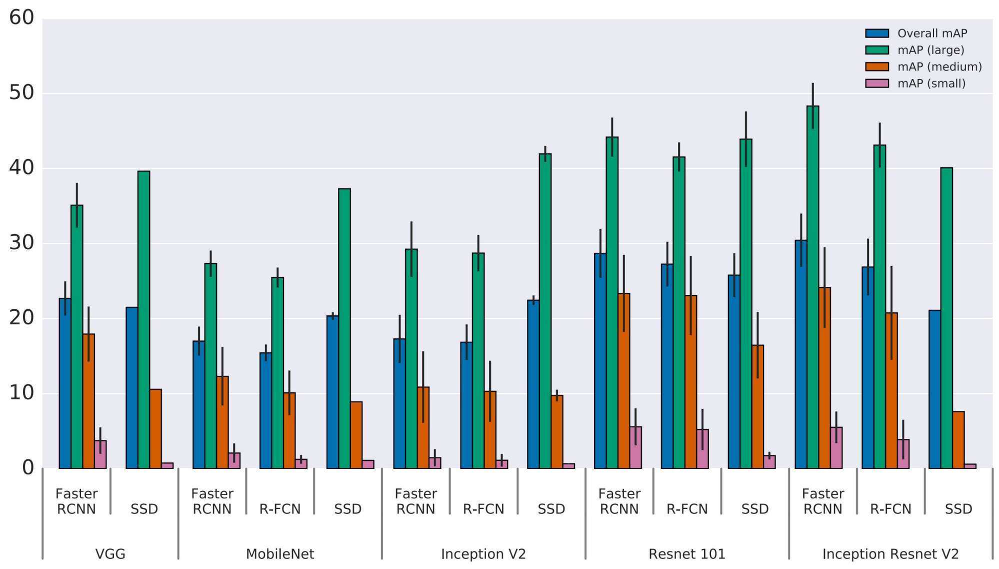


Figure 2.12: Speed and accuracy trade-off of different detection methods by object size and feature extractor [63].

In general, Faster R-CNN is more accurate while SSD is faster. Especially for the small object in images, Faster R-CNN outperforms SSD largely. As can be seen from Figure. 2.12, Faster R-CNN achieves more accurate detection result on all object sizes when a more complex ConvNet structure such as ResNet is used [30]. Some paper also conducts research on ship detection with different detection methods. In [4], Faster R-CNN is also used to detect 10 different categories of ships because of its accuracy. It achieves 0.947 mean average precision. In [64], SSD is implemented as 2D detectors to detect different kinds of ships. Their best model reaches 0.84 mean average precision. Though it is not fair to compare the performance of different models trained on a different dataset, Faster R-CNN can reach better performance than SSD. According to COCO dataset object [65] detection challenge, all 2D detection algorithm on the leaderboard is a Faster R-CNN based method.

Deep3DBox achieves the second place on KITTI orientation estimation evaluation benchmark among all non-anonymous methods. Compare with other 3D detection methods, SubCNN [15] proposes a complicated region proposal network structure and Mono3D [61] requires heavy pre-processing on input images. Although DeepMANTA [16] achieves a better result than Deep3DBox, it needs a 3D CAD model to perform 2D to 3D key points matching. The 3D CAD model is difficult to create. Deep3DBox does not have complex pre-processing and ConvNet structure. Furthermore, the training data for it is easy to obtain with a recording platform. Thus, Deep3DBox is a better choice for 3D information estimation of ships.

Chapter 3

Methodology

During the time of working on this project by our knowledge, there have been no publications based on pose estimation of ships. We are the first to develop a ConvNet based 2D and 3D detection method on ships. In this work, the Faster R-CNN model is first deployed and trained on different categories of ships. Then the 3D bounding box estimation algorithm is attached to the 2D detection result. Finally, a Kalman filter is designed to generate more accurate results.

In this chapter, we will discuss the methodological details behind all these approaches. An overview of network architecture will be proposed at first. After that, the used methods in 2D and 3D detection will be illustrated respectively. Finally, we will describe the Kalman filter and the employed dynamic model.

3.1 Algorithm Choice

Based on the comparison in Section 2.7, we decide to choose Faster R-CNN as our meta-architecture in this project. Consider the actual maritime traffic condition, the detection accuracy is much more important than the detection rate. Maritime traffic is not complex and changeable as road traffic, so high detection speed is not the key factor. Conversely, it is important to classify categories of different ships. Furthermore, vessels in the image captured by the autonomous ship while sailing is likely to be small. If these vessels can be detected accurately, effective steps can be made for navigation and decision making. Deep3DBox is chosen for 3D object detection.

3.2 Network Structure

The architecture of our ConvNet is illustrated in Figure 3.1. During training, each image is resized to equal width and height initially. Pre-propossing techniques such as random flip can be used to on images to make the ConvNet more robust. After that, modern ConvNet such as VGG-16 [29] or ResNet [30] are attached upon images to extract feature maps. In this project, the simple VGG-16 is used because of its simple structure and high performance. On the output feature maps produced by the last convolution layer, the region proposal network (RPN) runs spatially to generate candidate regions and boxes that may contain objects. Among all the proposed bounding boxes, a non-maximum suppression (NMS) algorithm is used to filter out all redundant boxes. These regions are then pooled to a fixed size by using region of interest (ROI) pooling introduced in [19]. Consequently, all selected regions are fed into a fully-connected network to classify different objects. The 2D ship detection method ends here.

After obtaining the predicted 2D bounding boxes, they are fed into another ConvNet to project 3D bounding box. The cropped images run through several convolution layers and three fully-connected layers to estimate orientation and dimensions. In order to predict the orientation angle, the ConvNet first identifies if the angle lies in a certain discrete rotation angle range and then regress the residual rotation with respect to the center of the angle range. After obtaining

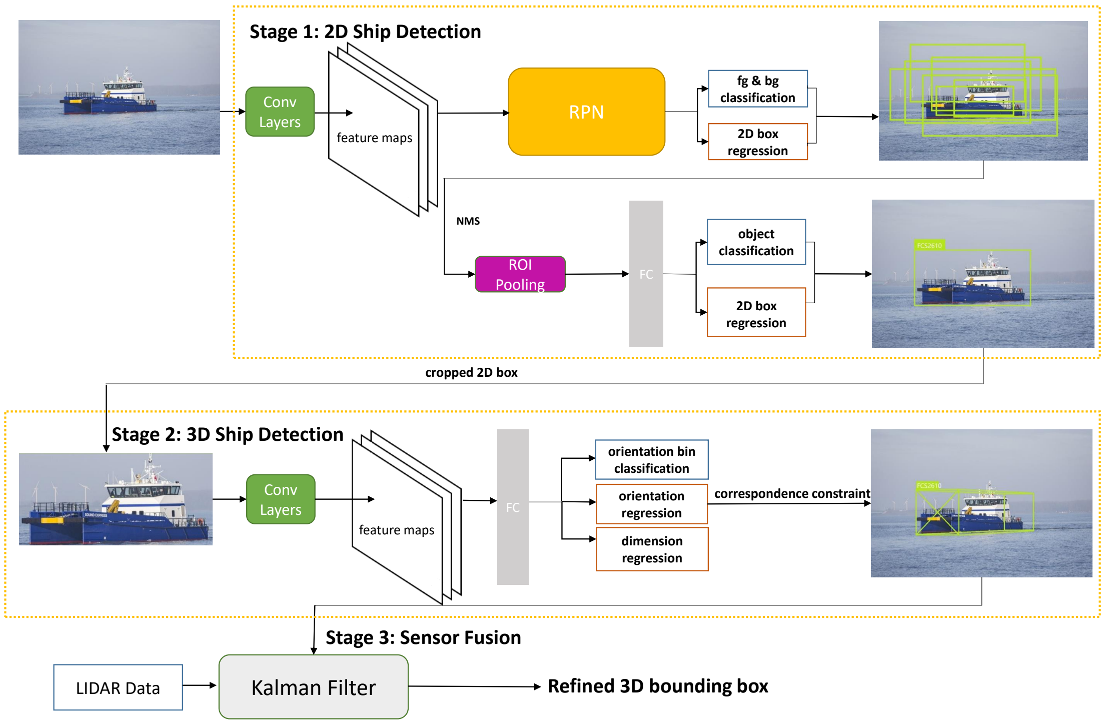


Figure 3.1: Overview of network structure. Our ConvNet based detection method can estimate 3D bounding boxes directly from monocular images.

accurate orientation and dimension from the ConvNet, a correspondence constraint is used to compute translation values based on camera matrix and predicted parameters. Finally, a Kalman Filter is designed to generate accurately predicted results by fusing information from different sensors together.

3.3 2D Ship Detection

The 2D ship detection approach is illustrated in Stage 1 in Figure 3.1. It can be separated into two sub-networks: RPN and a classification network. In this section, the used method for 2D detection is discussed.

3.3.1 Region Proposal Network

RPN, in general, consists of additional convolutional layers added on the top of feature maps that can simultaneously regress rectangular object regions and classify whether these regions belong to the foreground or background at each location. It outputs several rectangular object proposals with a score on each of it. Each score represents objectness of one proposal which means the rectangular region is foreground or background. In order to generate candidate regions, a small network with 3×3 spatial size slides over the obtained convolutional feature maps. This sliding window network reduces feature maps to lower dimension and several anchors are generated at the center of this sliding window. After that, two sibling fully connected layers are used to perform two tasks: bounding box regression and objectness classification. Note that two sibling 1×1 convolutional layers are used here as fully-connected layers in order to reduce the number of parameters. The RPN model is shown in Figure 3.2.

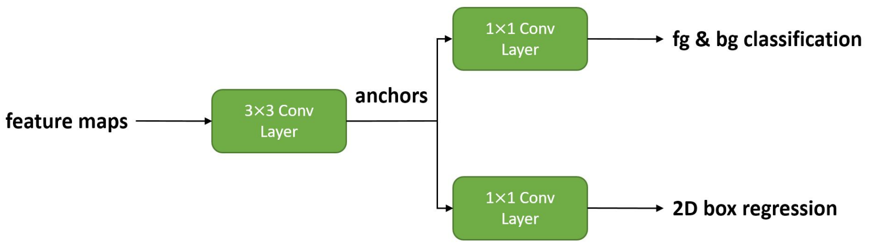


Figure 3.2: Architecture of the used RPN.

Anchors Generation

As the sliding window slides spatially through all feature maps, anchors are generated with respect to different size and ratio at the centre of each sliding window. In the original paper published by Faster R-CNN, three anchor sizes $[128, 256, 512]$ and three ratios $[1{:}1, 1{:}2, 2{:}1]$ are used for images. Note that not all anchors are used for objectness classification, anchors exceeding image boundaries are excluded. Then, an intersection over union (IoU) method is used to compute the overlap between anchors and ground truth bounding boxes in training process. The detailed explanation of IoU can be found in Section 4.2.1. The ground truth objectness score p^* is calculated via:

$$p^* = \begin{cases} 1, & \text{if} \quad \text{IoU} > 0.7 \\ 0, & \text{if} \quad \text{IoU} < 0.3 \\ \text{None}, & \quad \text{otherwise} \end{cases} , \tag{3.1}$$

giving that the IoU over 0.7 is marked as foreground and less than 0.3 is marked as background. This is because anchors have IoU larger than 0.7 with ground truth bounding box contains most part of the object [19]. For IoU between 0.3 and 0.7, these anchor are excluded because they are vague to identify their objectness. Note that during training if an object does not have any anchor with IoU larger than 0.7, the anchor with highest IoU will be marked as foreground. As a result, for each selected candidate anchor, the RPN outputs 2 scores to estimate probability of an object and 4 values to represent the bounding box coordinate. An illustration of how anchors are generated is given in Figure 3.3.

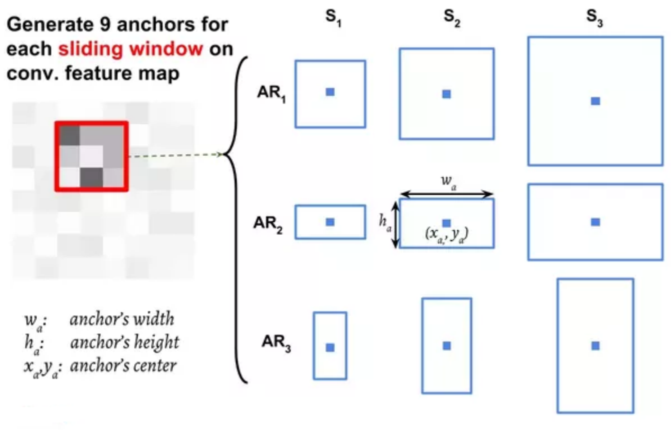


Figure 3.3: Illustration of anchors generated by RPN. At the center of each sliding window, 9 anchors in different ratios and scales are created [66].

RPN Loss Function

Since the RPN performs objectness classification and bounding box regression, the loss function for training it can also be separated into two parts. For objectness classification, the loss function $L_{cls}(p_i, p_i^*)$ is defined exactly the same as Equation 2.17 described in Section 2.3. For a single bounding box, it can be described by 4 parameters (x, y, w, h), where x, y are the center coordinate of the bounding box and w, h are the width and height of each box. Normally, regressing these 4 values directly will not yield accurate results according to many experiments [67, 18]. The offset values are regressed instead. Consider one ground truth bounding box (x, y, w, h) and a selected anchor (x_a, y_a, w_a, h_a) by RPN, the transformation from anchors to bounding box can be determined in these equations:

$$\begin{aligned} x &= t_x w_a + x_a, \\ y &= t_y h_a + y_a, \\ w &= e^{t_w} w_a, \\ h &= e^{t_h} h_a. \end{aligned} \tag{3.2}$$

The linear transformation is parameterized in 4 variables (t_x, t_y, t_w, t_h). These are our learnable parameters. During training, the RPN tries to minimize the difference between ground truth (t_x, t_y, t_w, t_h) and estimation $(t_x^*, t_y^*, t_w^*, t_h^*)$:

$$\begin{cases} t_x = \frac{x - x_a}{w_a}, & t_y = \frac{y - y_a}{h_a} \\ t_w = \log \frac{w}{w_a}, & t_h = \log \frac{h}{h_a} \end{cases}, \tag{3.3}$$

$$\begin{cases} t_x^* = \frac{x^* - x_a}{w_a}, & t_y^* = \frac{y^* - y_a}{y_a} \\ t_w^* = \log \frac{w^*}{w_a}, & t_h^* = \log \frac{h^*}{h_a} \end{cases}, \tag{3.4}$$

where x, x_a, x^* represents x coordinate of predict box, anchor box, and ground truth box respectively (same for y, w, h). The loss function is defined as smooth L_1 error:

$$\begin{aligned} L_{reg}(t_i, t_i^*) &= \sum_{i \in (x,y,w,h)} \text{smooth}_{L_1}(t_i - t_i^*), \\ \text{smooth}_{L_1} &= \begin{cases} 0.5x^2, & if \quad |x| < 1 \\ |x| - 0.5, & \text{otherwise} \end{cases}. \end{aligned} \tag{3.5}$$

Although the L_2 loss is more precise and better at prediction, it is more sensitive to outliers. It require careful tuning of learning parameter in order to prevent gradient explosion through back-propagation. In addition, normal L_1 loss $L_1 = |x|$ is not differentiable at 0, it can influence heavily on convergence of loss function. As a result, the smooth L_1 loss is used for bounding box regression. The total loss for RPN becomes:

$$L_{RPN}(p_i, t_i) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda_1 \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*). \tag{3.6}$$

where N_{cls} means the number of total anchors in one batch size, i is the index of each anchor, L_{cls} is defined as the softmax score with cross-entropy in Section 2.3, the ground truth label p^* is 1 if the anchor is foreground, otherwise the value is 0, λ_1 is a loss weighting parameter to ensure both cls and reg are approximately equally weighted. It is also a tuning parameter to ensure performance in both objectives. N_{reg} is the number of anchor locations. The term $p^* L_{reg}$ means the regression loss is activated only for anchor belongs to foreground ($p^* = 1$) and disabled for background anchors ($p^* = 0$).

3.3.2 Classification Network

The classification network is applied on the proposed feature maps from RPN and classifies them to a specific category through fully connected layers and softmax classifier. The bounding box regression is used again to obtain more accurate results.

Non-maximum suppression

After training RPN, it can successfully output rectangular proposals on images. However, many anchors could be classified as foreground by RPN. Hence, NMS is used to reduce the number of proposals. This method can be divided into the following steps:

1. Find the predicted proposal with highest score as foreground;
2. Compute the IoU between this bounding box and other predicted bounding boxes;
3. Eliminate all other bounding boxes with IoU larger than 0.7;
4. With all the left proposal regions, repeat from step one.

The NMS method can significantly reduce the number of proposal regions. In practical, we set the number of rectangular boxes after NMS to be approximately 300. After that, all these selected regions are feed into a classification network to perform the object classification task and bounding box regression once again for better results.

ROI pooling

The RPN can successfully generate candidate regions that contain objects. However, these regions are of different spatial sizes. The fully-connected requires uniform feature sizes to perform object classification. The ROI pooling is introduced to make non-uniform sizes proposals into small feature maps with a fixed size (normally 7×7). It firstly divides proposed regions into 7×7 different blocks. Among each block, a max pooling operation introduced in Section 2.2.3 is used to output feature maps in equal size. Then, all these equal size feature maps are fed into two fully-connected layers to perform classification.

Loss function

The loss function in the classification network is in principle the same as in RPN. However, instead of classifying a proposed region to foreground and background, it classifies the region to a certain object category. The loss function is chosen as follows:

$$L_{CLN}(o_i, t_i) = \frac{1}{N_{obj}} \sum_i L_{cls}(o_i, o_i^*) + \lambda_2 \frac{1}{N_{obj}} \sum_i L_{reg}(t_i, t_i^*), \tag{3.7}$$

where N_{obj} means the total number of objects in one batch size, o^* is the ground truth value for various objects, λ_2 is the loss weighting parameter. The total loss for 2D ship detection is computed by:

$$L = L_{RPN} + L_{CLN}. \tag{3.8}$$

After constructing the model, the ConvNet can be trained from end to end to generate 2D bounding boxes on images. The detail of the implementation is given in Section 4.3.1.

3.4 3D Ship Detection

After successfully detecting ships in the 2D case, the 3D ship detection ConvNet is developed. Stage 2 shows the necessary steps needed to predict 3D bounding boxes on ships. In this section, the detailed steps behind 3D detection are described.

3.4.1 Estimation of 3D Bounding Boxs

As described in Section 2.6.2, 9 variables $(x, y, z, \phi, \psi, \theta, w, h, l)$ need to be determined in order to fit a 3D bounding box on objects. Normally for autonomous vehicles, the object roll ϕ and pitch ψ are assumed to be zero to simplify the problem. This further reduces the unknown variables to 7. Before exploring deeply into the methodology, we first need to know how to project 3D points in real-world coordinate systems on images.

Camera Matrix

The camera matrix is a 3×4 matrix that describes the mapping of a camera from 3D points in the world to 2D points in an image. The choice of reference frame will be detailed in the next section. It can be decomposed into a product of extrinsic and intrinsic camera parameters. The extrinsic camera matrix describes how to convert 3D points in world coordinate to camera coordinate. It has two components: the rotation matrix R and a translation vector T. This matrix has the same structure as a rigid body transformation. The transformation can be described in the following way:

$$\begin{aligned} \begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} &= \begin{bmatrix} I & T \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} R & \mathbf{0} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}, \\ &= \begin{bmatrix} R & T \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}, \end{aligned} \tag{3.9}$$

where $[X_c\, Y_c\, Z_c]$ describes the 3D point in camera coordinates and $[X_w\, Y_w\, Z_w]$ describes it in world coordinates. The homogeneous coordinates allow us to separate R and T in two matrices. The intrinsic camera matrix transforms 3D camera coordinates to 2D homogeneous images coordinates. This perspective projection is parameterized by Hartley and Zisserman in [68]. It can be described by:

$$K = \begin{bmatrix} f & 0 & c_x \\ 0 & f & c_y \\ 0 & 0 & 1 \end{bmatrix}, \tag{3.10}$$

where f is the focal length, c_x and c_y are the coordiantes of the principal point offset which describes the location of camera ray on images with respect to the image origin. By combining intrinsic matrix K, rotation matrix R, and translation T, a 3D point in object coordinate reference $\mathbf{X} = [X_w\, Y_w\, Z_w\, 1]$ can be projected into image plane $\mathbf{x} = \begin{bmatrix} s \cdot x & s \cdot y & s \end{bmatrix}$. s is a scale variable in homogeneous coordinate:

$$\begin{bmatrix} s \cdot x \\ s \cdot y \\ s \end{bmatrix} = K \begin{bmatrix} R & T \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}. \tag{3.11}$$

Coordinate System, Orientation, and Translation

The coordinate system also needs to be chosen at first. We will choose a coordinate system that corresponds to the KITTI dataset and its publication [69]. The detailed description of KITTI can be found at Section 4.1.2. For an autonomous vehicle, it should be equipped with multiple cameras to perceive the surrounding environment. Furthermore, the coordinate system should be in a fixed condition so that the transformation can be conducted correctly. According to the publication [69, 62], for convenient use, the positive value of directions in both the camera frame and body frame for objects are defined as:

$$\begin{aligned} x &= right, \\ y &= down, \\ z &= forward. \end{aligned} \tag{3.12}$$

However, the origin point of the camera frame and object frame are different. The center of the camera frame is located at the top of the recording platform. In contrast, its body frame is defined as the center of the bottom for each object. For example, if a ship model faces x positive direction, its body frame is illustrated in Figure 3.6:

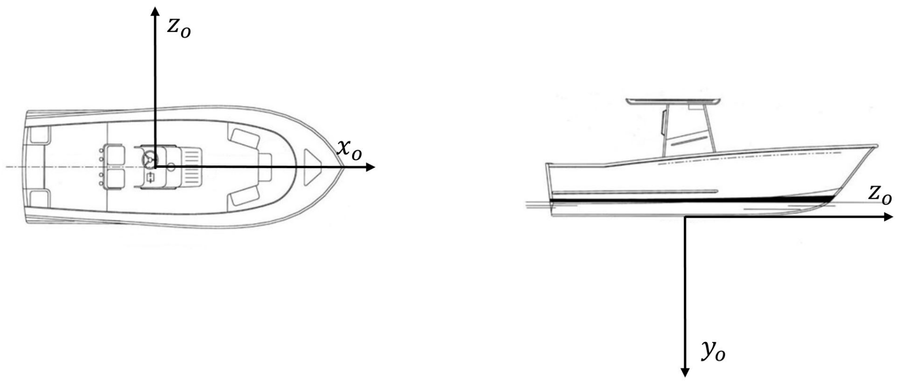


Figure 3.4: Left: Bird view of body frame for a single ship. Right: Front view of body frame.

After defining the coordinate system, the orientation yaw needs to be determined. There are two types of orientation normally used to describe objects, namely global orientation and local orientation. The global orientation describes the object rotation in world frame which remains unchanged while the object is moving straight. Conversely, local orientation is defined in camera coordinate which considers the ray from camera center through the cropped object center. A figure illustration is shown in Figure 3.5.

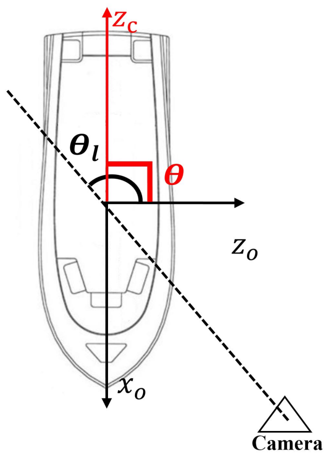


Figure 3.5: Illustration of local orientation θ_l and global orientation θ. The local orientation also considers the ray with respect to the camera.

In general, global orientation represents the object coordinate system with respect to the camera coordinate system. local orientation also considers the direction of camera ray.

If the intrinsic camera matrix is given, the global orientation can be obtained by combining local orientation and the crop center of each object instance.

$$\begin{aligned} &d = \sqrt{(x_o - c_x)^2 + (y_o - c_y)^2}, \\ &\theta_{ray} = \arctan(\frac{d}{f}), \\ &\theta = \theta_l + \theta_{ray}, \end{aligned} \tag{3.13}$$

where x_o and y_o are the centre point of an object on the image plane, d is the distance between ray and object centre on the image plane, θ_l and θ_{ray} are local orientation and ray angle respectively.

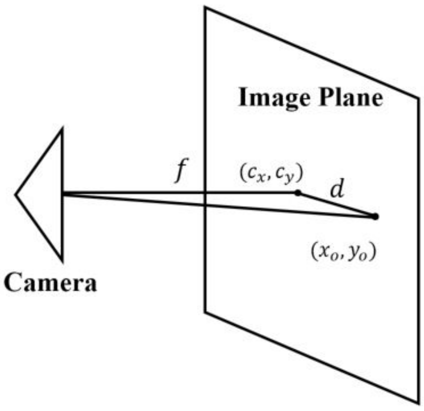


Figure 3.6: Illustration of how to compute θ_{ray}.

3D Bounding Box Construction

Once the translation between object and camera $[x, y, z]$ and the dimension of object $[w, h, l]$ are known. The eight coordinates of a 3D bounding box in body frame of one object $\mathbf{X}_{1\sim8} = \begin{bmatrix}\pm\frac{l}{2} & [0,h] & \pm\frac{w}{2}\end{bmatrix}$ can be transferred to image plane $\mathbf{x}_{1\sim8} = \begin{bmatrix}x_{1\sim8}, & y_{1\sim8}\end{bmatrix}$ by using Equation 3.9. Note that only the rotation yaw $R = \begin{bmatrix}\cos\theta & 0 & -\sin\theta \\ 0 & 1 & 0 \\ \sin\theta & 0 & \cos\theta\end{bmatrix}$ is considered here because roll ϕ and pitch ψ are considered close to 0. Some of the ground truth 3D bounding boxed are shown in Section 4.1.2 and 4.1.3.

3.4.2 Multi-Bin Networks

After knowing how to construct 3D bounding boxes. We need to determine parameters to regress. Orientation and dimensions are selected as the regression variables. Inspired by 2D object detection algorithms such as Faster R-CNN [12] and SSD [13], we also use a similar network structure named *MultiBin* to estimate 3D bounding boxes. The following sections will discuss this method.

Parameter Choices

Under the assumption that roll ϕ and pitch ψ are close to zero, we need to estimate 7 parameters to determine one 3D bounding box. The most influential one among these variables is yaw θ. It contains important 3D information and determines the heading of an object. Therefore, the local orientation θ_l is chosen as a parameter to regress. The local orientation value can be different even if the global orientation remains unchanged. As can be seen in Figure 3.7, the global rotation of a car is constant, but the local orientation angle changes at different positions. Hence, it is

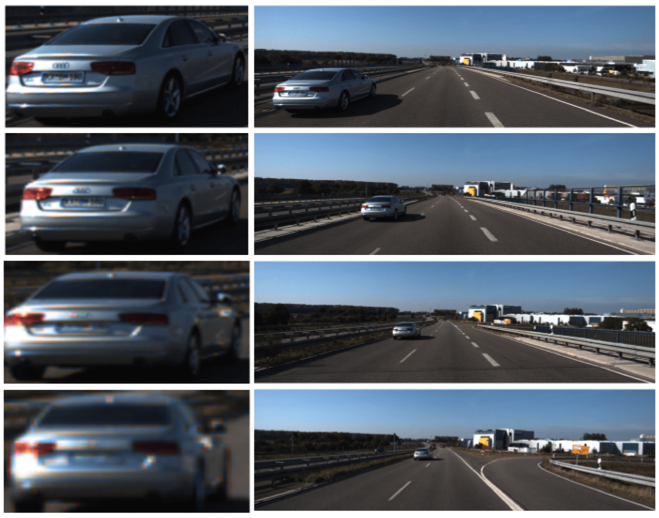


Figure 3.7: Left: cropped images of a car. Right: Images of the whole scene. The local orientation changes in different images while the global orientation remains the same [20].

reasonable and easy to regress local orientation based on camera images. Same for ships, normally for a sailing ship its roll and pitch angle are very small and not considered as an important factor. For ship navigation, the most influential factor is considered to be the yaw angle.

We choose to regress are dimensions $[w\,h\,l]$ rather than translation $[x\,y\,z]$. This is because the dimension values vary not much for a certain type of object in reality (e.g. tugs roughly have the same size). Furthermore, dimension parameters heavily depend on a particular subcategory that can most be recovered accurately if that subcategory can be classified correctly. In contrast, there is no regular pattern can be observed in three translation values x, y, and z based on the cropped object window. Consequently, dimensions and orientations are chosen as parameters to regress.

Multi-Bin Structure

Similarly to bounding box regression problem, estimate the orientation angle directly will not yield accurate results. Instead, the orientation angle is firstly divided into n overlapping bins. We then estimate both the confidence probability c_i that the predicted angle is involved in i^{th} bin and the angle offset which needs to be added to the ray of that bin to obtain the final local orientation. This approach divides the orientation regression problem into two sub-problems, namely bins classification and angle offset regression. A simple two bins model is shown in Figure 3.8.

The angle offset is represented by the cosine and sine value of the angle instead of the orientation value itself based on the loss function in next section. This results in 3 outputs for the orientation estimation which are confidence, cosine and sine value respectively. The cosine and sine values are obtained by applying L_2 normalization layer at the output of fully-connect layers. The exact network structure is shown in Figure 3.9. The fully-connected layers for regressing dimensions have 256 units and 512 units for both orientation branches.

Loss Function

The loss function for orientation regression is defined as the Euclidean distance between the ground truth angle offset $\Delta\theta_i$ and the estimated offset $\Delta\hat{\theta}_i$. For a single bin, the loss function is as follows:

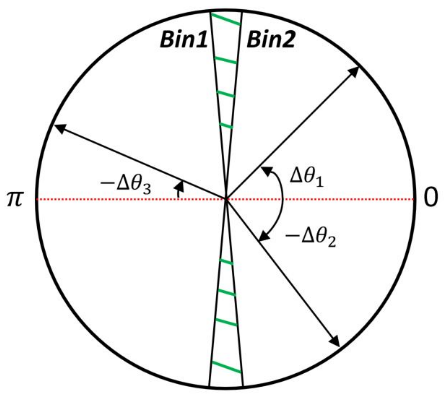


Figure 3.8: A simple $MultiBin$ model in two bins and with 10 degree overlapping, the region with green lines means the overlapping part, the red dotted line is the center ray of each bin, angles in anti-clockwise are considered to be a positive offset and in clockwise to be a negative offset.

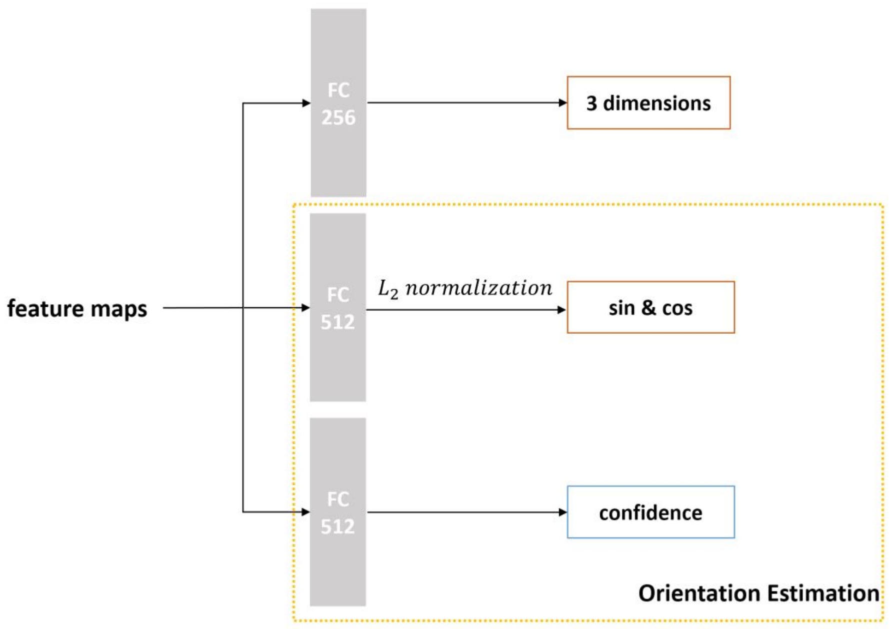


Figure 3.9: $MultiBin$ architecture for orientation and dimension estimation.

$$\begin{aligned} L_o &= \frac{1}{n}\sum_{i}^{n}[cos(\Delta\theta_i) - cos(\Delta\hat{\theta}_i)]^2 + [sin(\Delta\theta_i) - sin(\Delta\hat{\theta}_i)]^2, \\ &= 2 - \frac{2}{n}\sum_{i}^{n}[cos(\Delta\theta_i)cos(\Delta\hat{\theta}_i) + sin(\Delta\theta_i)sin(\Delta\hat{\theta}_i)], \end{aligned} \tag{3.14}$$

where n is the batch size. For multiple bins, the $cos(\Delta\theta_i)cos(\Delta\hat{\theta}_i) + sin(\Delta\theta_i)sin(\Delta\hat{\theta}_i)$ value at different bins n_θ are added together. In that case, the loss function results in:

$$L_o = 2 - \frac{2}{n}\sum_{i}^{n}\sum_{j}^{n_\theta}[cos(\Delta\theta_i)cos(\Delta\hat{\theta}_i) + sin(\Delta\theta_i)sin(\Delta\hat{\theta}_i)]_j. \tag{3.15}$$

The loss function for confidence is equal to the softmax score with cross-entropy (Section 2.3) of confidence in each bin. The eventual loss for local orientation θ is:

$$L_\theta = L_{conf} + \beta \times L_o, \tag{3.16}$$

where L_{conf} is the loss for confidence and β is loss weight for angle offset. After determining the orientation loss, we need to define the loss function for dimensions. The dimension loss is simply defined as L_2 loss for residual dimensions with respect to different training types in the dataset. It is computed as follows:

$$L_{dims} = \frac{1}{n}\sum_{i}^{n}(\Delta_D - \delta)^2, \tag{3.17}$$

where Δ_D is equal to the ground truth dimensions of each object D minus the average dimension $\bar{D}$ over one particular category. δ is the residual dimension values estimated by the network. The total loss for this ConvNet becomes:

$$L = \alpha \times L_{dims} + L_{conf} + \beta \times L_o, \tag{3.18}$$

where α is loss weight for dimensions. The loss weight is used to ensure different loss functions converging during the training procedure. It is determined empirically.

3.4.3 Correspondence Constraint

The dimensions and orientations can be regressed accurately after training the ConvNet. However, the translation value is still unknown which is essential to produce a 3D bounding box. An approach called correspondence constraint which considers the constraints between 2D detection box and 3D detection box is proposed to solve the translation problem analytically. Based on Equation 3.11, the following constraint can be formulated:

$$\begin{bmatrix} s\cdot x \\ s\cdot y \\ s \end{bmatrix} = K\begin{bmatrix} I & R\mathbf{X}_w \\ 0 & 1 \end{bmatrix}\begin{bmatrix} T \\ 1 \end{bmatrix}, \tag{3.19}$$

where I is identity matrix, $\mathbf{X}_w = [X_w\, Y_w\, Z_w]$ represents the eight coordinates of 3D bounding boxes in the body frame of each object, and $T = [T_x\, T_y\, T_z]$ is the unknown translation vector. The methodology assumes that the projection at a 3D bounding box on image plane should fit tightly into the 2D detection bounding box. This hypothesis means that the 2D bounding box parameters $[x_{min}\, x_{max}\, y_{min}\, y_{max}]$ can be the projection of any eight coordinates of the 3D bounding box in the body frame of each object $\begin{bmatrix}\pm dx/2 & [0, dy] & \pm dz/2\end{bmatrix}$. This results in $8^4 = 4096$ configurations. For example, x_{min} can be the projection of any points in 3D bounding box which results in:

$$\begin{bmatrix} s\cdot x_{min} \\ * \\ s \end{bmatrix} = \underbrace{K\begin{bmatrix} I & R\mathbf{X}_w \\ 0 & 1 \end{bmatrix}}_{M_{a_{3\times4}}}\begin{bmatrix} T \\ 1 \end{bmatrix}, \tag{3.20}$$

where $*$ means in the case of computing x_{min}, y value is not interested and considered. Similar for y_{min} part, the left part of Equation 3.20 will become $[*, s\cdot y_{min}, s]^\top$. If the underbraced part is denoted as M_a, the first and last row of this equation can be rewritten into the following equations in Python code.

$$\begin{cases} M_a\,[0,:]\begin{bmatrix} T \\ 1 \end{bmatrix} = s\cdot x_{min} \\ M_a\,[2,:]\begin{bmatrix} T \\ 1 \end{bmatrix} = s \end{cases}, \tag{3.21}$$

where : means all rows or columns are considered in the matrix. The above equation results in:

$$M_a\,[0,:]\begin{bmatrix}T\\1\end{bmatrix} = M_a\,[2,:]\begin{bmatrix}T\\1\end{bmatrix}\cdot x_{min}, \tag{3.22}$$

$$\underbrace{\{M_a\,[0,0:3] - M_a\,[2,0:3]\cdot x_{min}\}}_{A_{a_{1\times 3}}} T = \underbrace{M_a\,[2,3]\cdot x_{min} - M_a\,[0,3]}_{b_{a_{1\times 1}}}. \tag{3.23}$$

For a specific configuration vector $\mathbf{c} = [a, b, c, d]$, where $\mathbf{c}$ can contain any 4 of the 4096 configurations (e.g. a = $\begin{bmatrix}dx/2 & dy & dz/2\end{bmatrix}$). Two matrices A_a and b_a in Equation 3.23 can be expanded to 4 parameters $[x_{min}\, x_{max}\, y_{min}\, y_{max}]$ of the 2D bounding box. This results in the following equation:

$$\begin{aligned} A[r,:] &= M_{c[r]}[i,0:3] - M_{c[r]}[2,0:3]\cdot bbox[r], \\ b[r] &= M_{c[r]}[2,3]\cdot bbox[r] - M_{c[r]}[i,3], \end{aligned} \tag{3.24}$$

where bbox represents the four bounding box coordinates for one object, i indicates the projection is conducted in x or y coordinate, r is the row index. These parameters are listed in Table. 3.1.

c	a	b	c	d
bbox	x_{min}	y_{min}	x_{max}	y_{max}
i	0	1	0	1
r	0	1	2	3

Table 3.1: Parameters in Equation 3.24.

After computing matrix A and b, there are 4 different equations related to the 4 sides of the 2D bounding box. This results A in 4×3 matrix and b in 4×1 vector. The translation value for each configuration vector $\mathbf{c}$ can be calculated by solving:

$$\begin{aligned} AT &= b, \\ T &= (A^\top A)^{-1}A^\top b. \end{aligned} \tag{3.25}$$

As described before, there are 4096 configurations in total to solve the translation values for each object. This large number can be further reduced by some observations. Consider 3D and 2D bounding box in the figure below. In reality, ships are always upright, which means y_{min} and y_{max} can only be the projection of $[x_2, x_4, x_6, x_8]$ and $[x_1, x_3, x_5, x_7]$ respectively. This reduces the number of configurations to 1024. Furthermore, the roll and pitch angle for a sailing ship are both close to zero which makes x_{min} and x_{max} can only be the projection of $[x_5, x_6, x_7, x_8]$ and $[x_1, x_2, x_3, x_4]$, resulting in $4^4 = 256$ configurations. The correct translation can be selected from 256 configurations. However, the method to select the correct translation needs to be developed. In this project, we provide a sensor fusion technique to choose the right translation based on other sensor information.

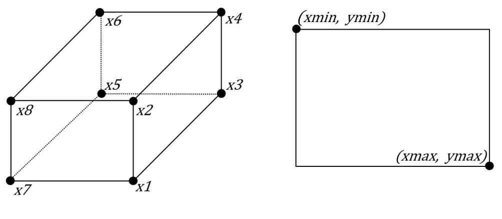


Figure 3.10: Left: 3D bounding box in body frame of an object. Right: 2D bounding box on images.

3.5 Sensor Fusion

After implementing 3D bounding box detection algorithm, the translation and orientation of objects can be successfully estimated from a monocular camera. However, in real autonomous shipping systems, multiple sensors are equipped onboard to obtain more robust and accurate information for tasks such as navigation and decision making. Sensor fusion [70] is a technique that combines sensory data derived from disparate sources such that the resulting information can be much more accurate than each source used individually. Note that dynamic mothion motion of real ships can be more complicated. In this project we focus on a generic method to show that the perception ability can be improved by fusing information together.

3.5.1 Kalman Filter for Sensor Fusion

The task of a sensor is to provide relevant information about a process variable by measuring the outside environment. Measurements from different sources can be noisy and inaccurate. A Kalman Filter [71, 72] is often used to remove noise from sensor signals and fuse data together in order to simultaneously estimate the smoothed values of position, orientation, and velocity [73].

Kalman Filtering: Due to the sampled nature of the information, we choose to consider a discrete time formulation of our filtering and sensor fusion tasks. The standard discrete-time Kalman Filter model is described by the following linear time-invariant state-space model:

$$\begin{cases} x_{k+1} & = Ax_k + Bu_k + w_k \\ y_k & = Cx_k + v_k \end{cases}, \tag{3.26}$$

where x_k is a vector that contains the state variable at time k, u_k describes the input to the system at time k, A is a non-singular matrix and B is a matrix mapping input to state, w is a random variable that represents the process noise, modeled as white noise with Gaussian distribution $\mathcal{N}(0, Q)$. y_k contains the sensor observation at time k, C is the extraction matrix between state and measurement, and v is the measurement noise modeled as white noise with Gaussian distribution $\mathcal{N}(0, R)$.

The Kalman filter can be separated into the following steps. A priori estimator $\hat{x}_{k+1|k}$ of the state x at time $k+1$ is computed by using the mathematical model at time k:

$$\hat{x}_{k+1|k} = A\hat{x}_{k|k} + Bu_k. \tag{3.27}$$

Then, the predicted error covariance matrix P at time $k+1$ and the Kalman gain K is calculated by:

$$P_{k+1|k} = AP_{k|k}A^\top + Q, \tag{3.28}$$

$$K_{k+1} = P_{k+1|k}C^\top(CP_{k+1|k}C^\top + R)^{-1}. \tag{3.29}$$

The estimated state can be updated by the measurment y_{k+1}:

$$\hat{x}_{k+1|k+1} = \hat{x}_{k+1|k} + K_{k+1}(y_{k+1} - C\hat{x}_{k+1|k}). \tag{3.30}$$

The new error covariace matrix $P_{k+1|k+1}$ is obtained by:

$$P_{k+1|k+1} = (I - K_{k+1}C)P_{k+1|k}(I - K_{k+1}C)^\top + K_{k+1}RK_{k+1}^\top. \tag{3.31}$$

where I is the identity matrix. After calculation of Equation 3.31 the iteration restarts with Equation 3.27 and $k = k + 1$. The initial state $\hat{x}_0$ is often estimated by using linear least squares. In case the initial measurement is not accurate or the noise assumptions are only approximate, it can cost more time for the estimation error to converge.

3.5.2 Dynamic Model

In order to perform sensor fusion, the state variables are constructed to represent a so-called measurement model. We want to fuse information from different sensors to obtain accurate orientation and translation in x and z coordinates of a certain object. Hence, the dynamic model of a moving object with observed positions and pose can be described as follows:

$$\begin{cases} \theta_{k+1} &= \theta_k + \Delta t\omega_k + \frac{1}{2}\Delta t^2 a_\omega \\ x_{k+1} &= x_k + \Delta t u_k + \frac{1}{2}\Delta t^2 a_u \\ z_{k+1} &= z_k + \Delta t v_k + \frac{1}{2}\Delta t^2 a_v \\ \omega_{k+1} &= \omega_k + \Delta t a_\omega \\ u_{k+1} &= u_k + \Delta t a_u \\ v_{k+1} &= v_k + \Delta t a_v \end{cases}, \tag{3.32}$$

where θ, x, and z are orientation, translation in x and z direction. ω, u, and v are velocity of orientation and translation. a_ω, a_u, and a_v are accelerations of ω, u, and v. Δt is the time interval of the observation.

Since accelerations are unknown and depend on external forces and disturbances, we can add these variables to the process noise component. This gives the following state space model:

$$\begin{bmatrix} \theta_{k+1} \\ x_{k+1} \\ z_{k+1} \\ \omega_{k+1} \\ u_{k+1} \\ v_{k+1} \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & 0 & 0 & \Delta t & 0 & 0 \\ 0 & 1 & 0 & 0 & \Delta t & 0 \\ 0 & 0 & 1 & 0 & 0 & \Delta t \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}}_{\text{A}} \begin{bmatrix} \theta_k \\ x_k \\ z_k \\ \omega_k \\ u_k \\ v_k \end{bmatrix} + \underbrace{\begin{bmatrix} \frac{1}{2}\Delta t^2 a_\omega \\ \frac{1}{2}\Delta t^2 a_u \\ \frac{1}{2}\Delta t^2 a_v \\ \Delta t a_\omega \\ \Delta t a_u \\ \Delta t a_v \end{bmatrix}}_{w}, \tag{3.33}$$

where w is process noise with zero mean and covariance matrix Q. The vector w can be decomposed into two matrices G and a. Matrix G only contains constant variable Δt and matrix a contains the random acceleration components:

$$w = \underbrace{\begin{bmatrix} \frac{1}{2}\Delta t^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^2 \\ \Delta t & 0 & 0 \\ 0 & \Delta t & 0 \\ 0 & 0 & \Delta t \end{bmatrix}}_{\text{G}} \underbrace{\begin{bmatrix} a_\omega \\ a_u \\ a_v \end{bmatrix}}_{\text{a}} = Ga. \tag{3.34}$$

Based on the noise vector w, the covariance matrix Q can be determined. It is defined in the following equation:

$$Q = E[ww^\top] = E[Gaa^\top G^\top], \tag{3.35}$$

where E corresponds to the expectation operator. Matrix G can be lifted out of the expectation computation because it does not contain random variables:

$$Q = GE[aa^\top]G^\top = G \begin{bmatrix} \sigma_{a_\omega}^2 & 0 & 0 \\ 0 & \sigma_{a_u}^2 & 0 \\ 0 & 0 & \sigma_{a_v}^2 \end{bmatrix} G^\top, \tag{3.36}$$

where σ_{a_ω}, σ_{a_u}, and σ_{a_v} are the variance of a_w, a_u, and a_v respectively. a_w, a_u, and a_v are assumed to be uncorrelated.

The covariance matrix Q results in:

$$Q = \begin{bmatrix} \frac{1}{4}\Delta t^4 \sigma_{a_\omega}^2 & 0 & 0 & \frac{1}{2}\Delta t^3 \sigma_{a_\omega}^2 & 0 & 0 \\ 0 & \frac{1}{4}\Delta t^4 \sigma_{a_u}^2 & 0 & 0 & \frac{1}{2}\Delta t^3 \sigma_{a_u}^2 & 0 \\ 0 & 0 & \frac{1}{4}\Delta t^4 \sigma_{a_v}^2 & 0 & 0 & \frac{1}{2}\Delta t^3 \sigma_{a_v}^2 \\ \frac{1}{2}\Delta t^3 \sigma_{a_\omega}^2 & 0 & 0 & \Delta t^2 \sigma_{a_\omega}^2 & 0 & 0 \\ 0 & \frac{1}{2}\Delta t^3 \sigma_{a_u}^2 & 0 & 0 & \Delta t^2 \sigma_{a_u}^2 & 0 \\ 0 & 0 & \frac{1}{2}\Delta t^3 \sigma_{a_v}^2 & 0 & 0 & \Delta t^2 \sigma_{a_v}^2 \end{bmatrix}, \tag{3.37}$$

The covariance matrix of the state variables also needs to be initialized. Normally the initial value of is determined by the covariance of each variable which leads to:

$$P = \begin{bmatrix} \sigma_\theta^2 & 0 & 0 & 0 & 0 & 0 \\ 0 & \sigma_x^2 & 0 & 0 & 0 & 0 \\ 0 & 0 & \sigma_y^2 & 0 & 0 & 0 \\ 0 & 0 & 0 & \sigma_\omega^2 & 0 & 0 \\ 0 & 0 & 0 & 0 & \sigma_u^2 & 0 \\ 0 & 0 & 0 & 0 & 0 & \sigma_v^2 \end{bmatrix}, \tag{3.38}$$

where σ_θ, σ_x, σ_y, σ_ω, σ_u, σ_v are the variance of 6 state variables respectively commonly obtained empirically. Note that the covariance value can also be determined based on the certainty of different variables. More uncertainty results in a larger value.

After obtaining the covariance matrix P, the extraction matrix C needs to be decided. The orientation and location are measured variables here. Thus matrix C for each sensor is as follows:

$$C_1 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}, \cdots, C_n = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}. \tag{3.39}$$

By combining extraction matrices from different sensors together, the C matrix is equal to:

$$C = \begin{bmatrix} C_1 \\ \vdots \\ C_n \end{bmatrix}. \tag{3.40}$$

The final step is to determine the covariance matrix of measurement R for each interested object. However, for each object and sensor the matrix R is different. The detail of how to determine matrix R in a specific case is illustrated in Section 4.3.3.

Chapter 4

Implementation and Results

In this chapter, the implementation details and results evaluation are given. In order to train and evaluate the performance of ConvNet, various datasets for 2D ship detection, 3D ship detection, and sensor fusion are used and developed. After that, some evaluation metrics are provided to measure the accuracy of detection and sensor fusion results. The steps of how to train the constructed ConvNet are provided then. In the end, we present the results of our method on 2D and 3D ship detection and the proposed sensor fusion technique.

4.1 Datasets

Since deep learning is a data-driven approach, datasets become one of the most important parts to train and test a model. It can be broadly divided into two parts: training dataset and test dataset. The training dataset is used to tune hyper-parameters. During training, the ground truth for each example is given. The ConvNet tries to minimize loss function by updating parameters through the optimization algorithm. The test dataset is used to evaluate the performance of ConvNet. In test time, the input data goes through the model and the ConvNet will predict a series of relevant numbers. It is then compared with the ground truth label to show the performance of ConvNet. Due to the number of parameters in ConvNet can easily reach millions, a large number of dataset is needed to optimize these parameters in the training process. The success of a deep Learning project depends highly on the quality of dataset. Throughout this project, four datasets are used to evaluate the performance of each methodology. Since the lack of dataset in real marine traffic scene, we manually build a dataset for 2D and 3D ship detection. Furthermore, some open sourced dataset from the Internet is also used in this project. We now describe the label information and images in each dataset.

4.1.1 2D Ship Detection Dataset

For the purpose of autonomous sailing, there is no available dataset present. Annotate 2D bounding box manually of a larger number of images can be extremely laboursome and time-costly. Thus, we decided to create a small dataset that contains three different types of ships to train the detector. Specifically, we downloaded 639 images from Damen shipyard media library and these images were from three specific types of ships: ASD3212, FCS2610, and SPA4207. 570 images have been separated into the training set and the rest have been used to form the test set. We then used an open-sourced software labelImg [74] to annotate the locations of different kinds of ships on the images. Some visualization of this dataset is shown in Figure 4.1.

4.1.2 KITTI

The KITTI dataset was created for autonomous driving case. It was first introduced by Karlsruhe Institute of Technology in 2012 [75]. It contains images extracted from video sequences recorded

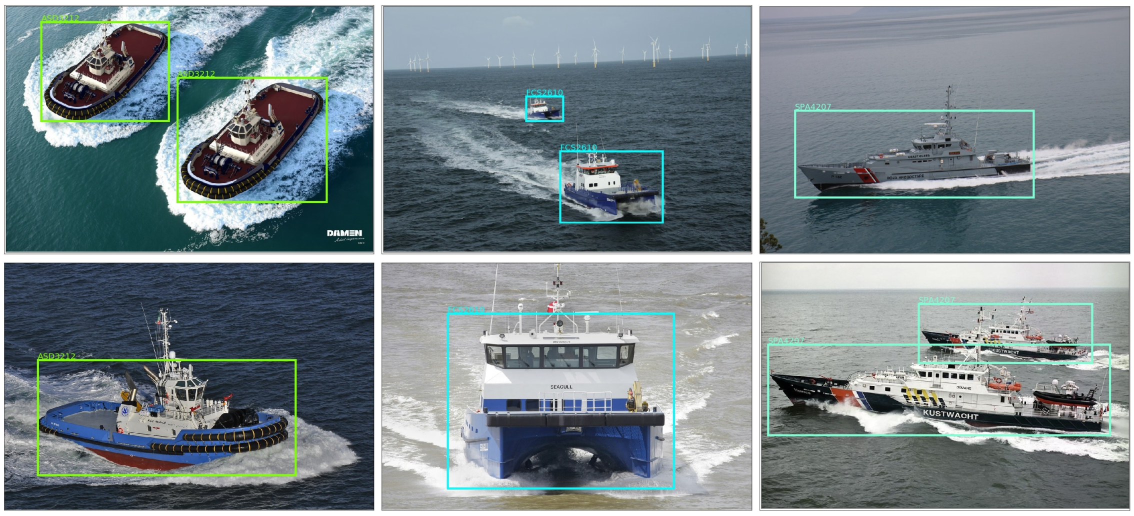


Figure 4.1: Sample images in the constructed 2D ship dataset.

in the city of Karlsruhe. The object detection benchmark in KITTI consists of 7481 training images and 7518 test images, with 80256 labeled objects in total. The images in KITTI dataset have a relatively high resolution (1240 × 375). Only the labels for the training set are provided as there is an active competition on the test dataset. Furthermore, the labels contain accurate information such as orientation, bounding box, and dimensions. The label information is in the following format:

type truncation occlusion alpha bbox dimensions location rotation_y

where type illustrate the type of objects such as car, pedestrian, or cyclist, truncation refers to the object leaving image boundaries, occlusion describes if the object is occluded by other objects, alpha is the local orientation, bbox means the 4 coordinates for a 2D bounding box, dimensions are the height, width, and length for objects in meters, location contains the 3 location parameters, rotation_y is the global orientation of an object.

Due to the lack of available similar dataset for autonomous sailing, the KITTI dataset is used to train and evaluate the 3D detection part because of the full 3D information it contains. Only cars, pedestrian, and cyclist are extracted from the labeled dataset. Some of the labels in KITTI dataset and its ground truth 3D bounding box are illustrated in Figure 4.2.

4.1.3 3D Ship Detection Dataset

Based on the KITTI dataset, a similar dataset with a toy ship is built to evaluate the performance of 3D detection. We first calibrated a web camera to obtain the camera intrinsic matrix. After that, we measured the dimensions and locations of the toy ship with respect to the camera by rulers. As a result, 88 images are recorded and labeled manually and 72 of them are used to train the Convnet. Some pictures from the ground truth training dataset are shown in Figure 4.3.

4.1.4 Sensor Fusion Dataset

During the period of this research work, no dataset and information from other sensors are provided. Thus, an imaginary dataset based on ground truth information is created in order to evaluate the performance of sensor fusion method. It is more realistic to implement this method on continuous frames so that the KITTI raw dataset [76] is used to perform sensor fusion. The KITTI dataset uses Velodyne HDL-64E laser scanner to generate lidar information. It is a 3D lidar sensor and has a 120m detection range. It also has ±2 detection accuracy. Based on this information, an imaginary lidar data is created. We add 2° orientation error and $30cm$ location

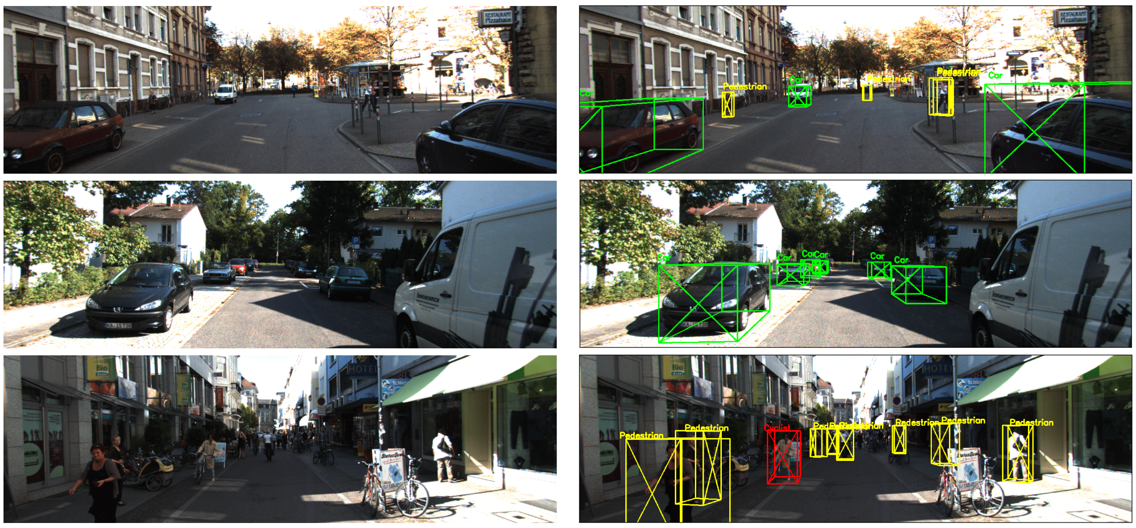


Figure 4.2: Sample images in the KITTI autonomous driving dataset.

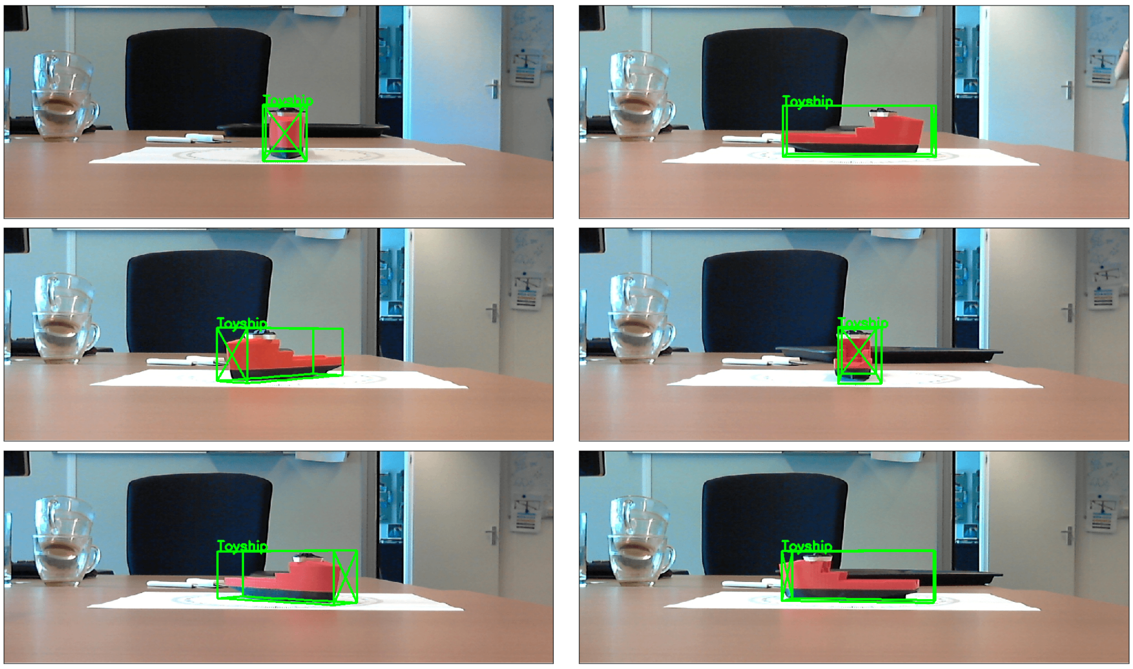


Figure 4.3: Sample images in the 3D toy ship dataset.

measurement error on the ground truth data to generate lidar data information with uniform distribution. Consequently, each file in the sensor fusion dataset results in the following format:

Timestamp obj0 angle* x* z* obj0 angle^1 x^1 z^1 obj0 angle^2 x^2 z ^{2} obj1 $\cdots$

where * means data from ground truth, ^{1} and ^{2} means data from camera and lidar respectively.

4.2 Evaluation

In this section, the evaluation metrics for 2D detection and 3D detection will be discussed respectively. Evaluation metrics are designed to measure the correctness of ConvNet's performance. For each of them, we are going to give the essential details and the method of computation.

4.2.1 2D Metrics

Intersection over Union

In the object detection field, intersection over union (IoU) measures the similarity of two bounding boxes. Before computing the IoU, we first calculate the intersection and union part separately. Consider there are two 2D bounding boxes $b1 = (x_1, y_1, x_2, y_2)$ and $b2 = (x_3, y_3, x_4, y_4)$. The intersection and union are computed seperately:

$$\begin{aligned} w &= \min(x_2, x_4) - \max(x_1, x_3), \\ h &= \min(y_2, y_4) - \max(y_1, y_3), \\ I &= wh. \end{aligned} \tag{4.1}$$

The union is computed by:

$$\begin{aligned} \text{aera}_{b_1} &= (x_2 - x_1)(y_2 - y_1), \\ \text{aera}_{b_2} &= (x_4 - x_3)(y_4 - y_3), \\ U &= \text{aera}_{b_1} + \text{aera}_{b_2} - I. \end{aligned} \tag{4.2}$$

The IoU can be computed as:

$$\text{IoU}(b_1, b_2) = \frac{I}{U}, \tag{4.3}$$

where $A()$ represents area and b_1, b_2 are two bounding boxes. Normally for object detection, people require $IoU \geq 0.5$ to accept a detection as a correct one. Later we will show an evaluation metric compares IoU from 0.5 to 0.95.

Precision and Recall

When evaluating models for binary classification on a dataset consisting of positive and negative samples, usually four types of data are defined: true positive (TP), true negative (TN), false positive (FP) and false negative (FN), whereas true and false refer to whether the positives or negatives are correctly classified by the model, see Figure 4.4. The following image shows the difference between these four parameters.

For multi-class classification, the negative samples of one category refer to all the other classes. With the numbers of TP, TN, FP, and FN, we adopt the classic precision-recall metrics defined as:

$$\begin{aligned} precision &= \frac{TP}{TP + FP}, \\ recall &= \frac{TP}{TP + FN}. \end{aligned} \tag{4.4}$$

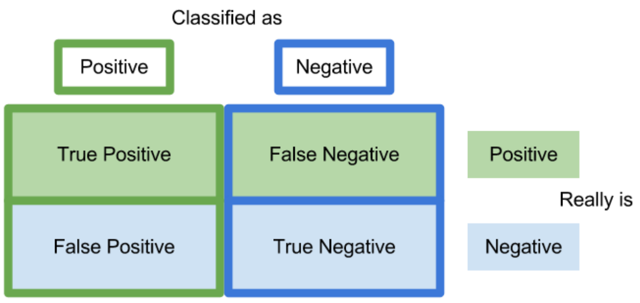


Figure 4.4: An image illustration of TP, TN, FP, and FN

In object detection, these parameters can be further explained in detail. A TP result means a detected bounding box with IoU larger than the threshold while FP means the detected bounding box with IoU smaller than a threshold (e.g. 0.5). FN means a ground truth bounding box is not detected. The precision (P) means the ability of a model to identify the relevant objects. It represents the percentage of correct positive predictions. The recall (R) is the ability of a model to find all relevant cases. It is the true positive detected objects among all ground truth bounding boxes. Equation 4.4 can be rewritten as:

$$\begin{aligned} precision &= \frac{TP}{\text{all detections}}, \\ recall &= \frac{TP}{\text{all ground truths}}. \end{aligned} \tag{4.5}$$

Average Precision

Average precision (AP) is adopted and used in every dataset benchmarks such as KITTI, PASCAL VOC [56], and COCO [65] to measure the accuracy of object detectors. It is the average of the maximum precision at different recall values between 0 and 1. It is computed by:

$$AP = \int_0^1 p(r)dr, \tag{4.6}$$

where $p(r)$ is the precision at recall r. The normal way to calculate AP is to sample the precision at 11 recall levels (0, 0.1, $\cdots$, 1.0), interpolate it and compute the mean value.

$$AP = \frac{1}{11} \sum_{r\in(0,0.1,\cdots,1)} p(r). \tag{4.7}$$

To evaluate the performance of the 2D ship detector, the COCO evaluation metric is used which measure 10 different AP from IoU at 0.5 to IoU at 0.95.

4.2.2 3D Metrics

In many open sourced dataset, the commonly used orientation evaluation metric is Average Orientation Similarity (AOS), which is a multiplication of 2D detection AP and average cosine distance similarity of orientation. Due to the lack of dataset, the 2D and 3D detection results are evaluated separately in our case. The 2D detection precision is removed from the metric. Therefore, an orientation Score (OS) is implemented to measure the performance of the orientation estimation result. The definition of OS is as follows:

$$\text{OS} = \frac{1}{n} \sum_{i}^{n} \frac{1 + \cos(\Delta_{\theta_i})}{2}, \tag{4.8}$$

where n is the number of objects in the validation dataset, Δ_θ is the difference between estimated and ground truth orientation angle. To evaluate the estimation result of dimensions and translations, Mean Squared Error (MSE) metric is used. The evaluation equations for both parameters are as follows:

$$\begin{aligned} MSE_d &= \frac{1}{n}\sum_i^n \frac{\Delta_w^2 + \Delta_h^2 + \Delta_l^2}{3}, \\ MSE_t &= \frac{1}{n}\sum_i^n \frac{\Delta_{T_x}^2 + \Delta_{T_y}^2 + \Delta_{T_z}^2}{3}, \end{aligned} \tag{4.9}$$

where dimensions and translations are $[w\,h\,l]$ and $[T_x\,T_y\,T_z]$ respectively.

4.3 Implementation Details

4.3.1 2D Detection

For 2D detection, the VGG-16 structure is used as the ConvNet to extract feature maps from images. The ConvNet takes full images as input and reshapes all of them to a fixed size (1024 $\times$ 600). Consider that in real marine traffic, a ship in captured images can be very small. Another anchor with size 64 is added to improve the detection accuracy for small ships in images. This results in totally 12 anchors generated at one location on feature maps. The loss weights for RPN and classification network are both set as 2 to make RPN loss function and classification loss function approximately equal. The RPN and classification network are trained alternatively. The foreground predicted by RPN needs to have an IoU over 0.7 compared with the ground truth bounding box in order to successfully train classification network. The model is trained with stochastic gradient descent (SGD) with a fixed learning rate $2e^{-3}$ and momentum of 0.9 is used to accelerate the training process. The training batch size is set to 1 due to the limitation of graphic card capability. In total we train 245 epochs for this model and the pre-trained model from Tensorflow model zoo is used to initialize the model weights. The model is trained locally on a GTX 960M graphic card with 2GB memory size. It takes about ten hours to train the model.

4.3.2 3D Detection

For 3D detection on KITTI, the ground truth 2D bounding box is used and each is cropped and resized to 224 $\times$ 224 based on the images in the ImageNet dataset [10]. The VGG-16 without its fully-connected layers is implemented to extract features from all cropped images. The $MultiBin$ module is added to regress dimensions and orientations. The number of bins is set to 2 based on the best result achieved in the paper [20]. The overlapping is chosen to be 10 percent. In addition, Dropout [36] and LeakyReLU [24] are used instead to increase the network performance. To simplify the procedure, only fully visible objects are utilized for training. All training images are randomly flipped to make the network more robust. The network is trained with SGD and momentum of 0.9. The learning rate is fixed to $1e^{-4}$. The training process is carried out on Amazon Web Service (AWS) with a single NVIDIA K80 GPU. It ran for 25 epochs with a batch size of 8 and the best model are chosen by minimal validation loss. In addition, the loss weight are set to $\alpha = 4$ and $\beta = 10$ respectively. This is because we found the orientation loss difficult to minimize during training time. Since the ground truth annotations for KITTI test set are not released, the training images are split into a training set (90 percent) and validation set.

For 3D detection for the toy ship, the trained weights on KITTI are used to initialize the model. The batch size is changed to 1 and the training procedure running for 1 epoch. The remaining settings are the same as 3D detection on KITTI.

4.3.3 Parameter Choices in Kalman Filter

As described in Section 4.1.4, the KITTI raw dataset *2011_09_26_drive_0056* is chosen to validate performance of Kalman Filter. We first ran 3D detection on this dataset with continuous frames. After that, we fuse Lidar information and camera information together. The predicted error covariance matrix P is determined at first. The value is determined based on the certainty of different variables. The orientation and location have more certainty than the velocity value since they can be measured directly from sensors. The initial covariance matrix P is determined empirically as the following equation, its effect will diminish as time progresses:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 100 & 0 & 0 \\ 0 & 0 & 0 & 0 & 100 & 0 \\ 0 & 0 & 0 & 0 & 0 & 100 \end{bmatrix}. \tag{4.10}$$

Note that increasing values in P will make the Kalman Filter trust the mathematical model more. It will result in smoother and more linear estimation result since the model is linear in our case. However, it may output inaccurate fusion result. Decrease values in P can cause the Kalman Filter trust measurements more. It can output more accurate result, but the estimation curve may fluctuate.

The three variance value σ_{a_ω}, σ_{a_u}, and σ_{a_v} in the process noise covariance matrix Q are all determined as 0.1. The tuning effect of Q is the same as tuning P.

We find the estimation results for a certain car $Car0$ to be fairly accurate. However, the result for $Cyclist0$ has been found below expectation. Therefore, the covariance matrix R for two different objects are determined in the following way.

$$\begin{aligned} R_{1_{car}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, & R_{2_{car}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \\ R_{1_{cyclist}} &= \begin{bmatrix} 10 & 0 & 0 \\ 0 & 10 & 0 \\ 0 & 0 & 10 \end{bmatrix}, & R_{2_{cyclist}} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}, \end{aligned} \tag{4.11}$$

where R_1 and R_2 are covariance matrix for camera and lidar respectively. Due to the inaccuracy of camera information for cyclist, the value in R_1 is higher to penalize the detection result. By combining R_1 and R_2 in the following way, we can follow from Equation 3.27 to update all parameters.

$$R = \begin{bmatrix} R_1 & 0 \\ 0 & R_2 \end{bmatrix}. \tag{4.12}$$

4.4 Results

4.4.1 2D Detection Results

As illustrated before, we use the COCO evaluation metric to test our model's performance. Tabel 4.1 and Figure 4.5 shows the 2D detection result on three types of ships. In the table, it can be found that the AP from IoU 0.5 to 0.95 is 0.728. Hence, the 2D detection model achieves accurate result. Specially for IoU at 0.5, the score for AP reaches 0.991. From the figure, we can see that the value of AP is around 0.95 for IoU between 0.5 and 0.7. After IoU of 0.75, it drops rapidly. This phenomenon is logical, since it is difficult to predict bounding box exactly the same as ground truth bounding box.

Average Precision (AP)	[IoU = 0.50:0.95 | area = all | maxDets = 100] = 0.728
Average Precision (AP)	[IoU = 0.50 | area = all | maxDets = 100] = 0.991
Average Precision (AP)	[IoU = 0.75 | area = all | maxDets = 100] = 0.933

Table 4.1: Detection evaluation results use COCO dataset metrics.

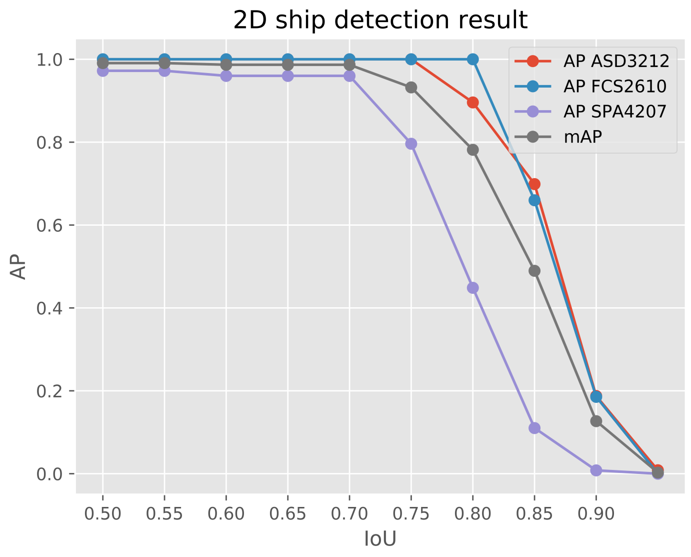


Figure 4.5: An image illustration of 2D detection result.

4.4.2 3D Detection Results

The evaluation result of 3D detection on KITTI is summarized in Table 4.2. As can be seen from the table, the *MultiBin* model can predict the orientation value accurately. The OS value can be converted to orientation error (OE) by using $arccos(2\times OS-1)$ formula. This results in 2.7°, 10.0°, and 18.0° angle errors on car, cyclist, and pedestrian respectively. Compare with other methods, the original paper [20] achieves 3.4° angle error on car, SubCNN [15], Mono3D [61], and 3DOP [14] results in 4.5°, 13.7°, and 15.1° angle error on car respectively. The *MultiBin* structure also succeeds in the prediction of dimensions. The MSE of dimensions is in relatively small value. In addition, the translation values selected from 256 configurations are also reasonable. The reason why MSE value for translations are slightly higher is that it is calculated based on estimated dimensions and orientation, which further increase in the error value.

Type	OS	MSE_d	MSE_t	OE
Car	0.9994	0.0078	1.8363	2.7°
Cyclist	0.9924	0.0066	1.6258	10.0°
Pedestrian	0.9754	0.0103	0.2409	18.0°
Toy Ship	0.9899	/	/	11.5°

Table 4.2: 3D detection results on both the KITTI and toy ship datasets.

As we described before, we only evaluate the orientation estimation result on toy ship dataset. The result is also shown in Table 4.2. As can be seen from the table, the orientation result is slightly worse than it on the KITTI dataset. This is because the number of training images is not enough and the measurement by hand in training dataset is not accurate enough.

4.4.3 Sensor Fusion Results

We ran the sensor fusion method on car and cyclist for 18 and 12 frames respectively. The orientation and location estimation results are shown in Table 4.3, Figure 4.6 and 4.7. As can be seen from the table and plots, the Kalman Filter can significantly reduce the prediction result by fusing information from camera and Lidar together. The estimation result for car has an offset about 0.02 rad, the method managed to reduce the offset by fusing Lidar information with visual estimation result together. Furthermore, the detection result for cyclist based on camera has 11.46° angle error which is below expectations. As described in Section 4.3.3, we decide to trust the information from Lidar more. As a result, the detection accuracy for cyclists is largely improved especially for the orientation estimation error reducing from 11.5 degree to only 1.5 degree.

	Car0			Cyclist		
Sensor	Angle Error	MSE_x	MSE_z	Angle Error	MSE_x	MSE_z
Camera	1.42°	0.0073	0.10	11.46°	1.27	7.40
Fusion	0.63°	0.008	0.007	1.51°	0.02	0.095

Table 4.3: Sensor fusion results on KITTI raw dataset.

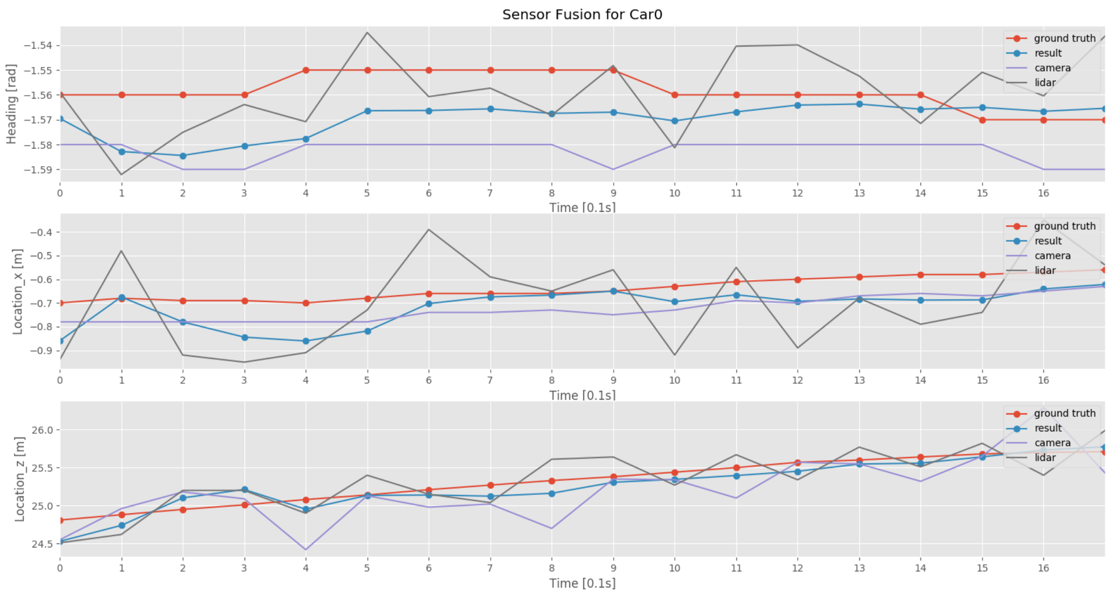


Figure 4.6: Sensor Fusion result for Car.

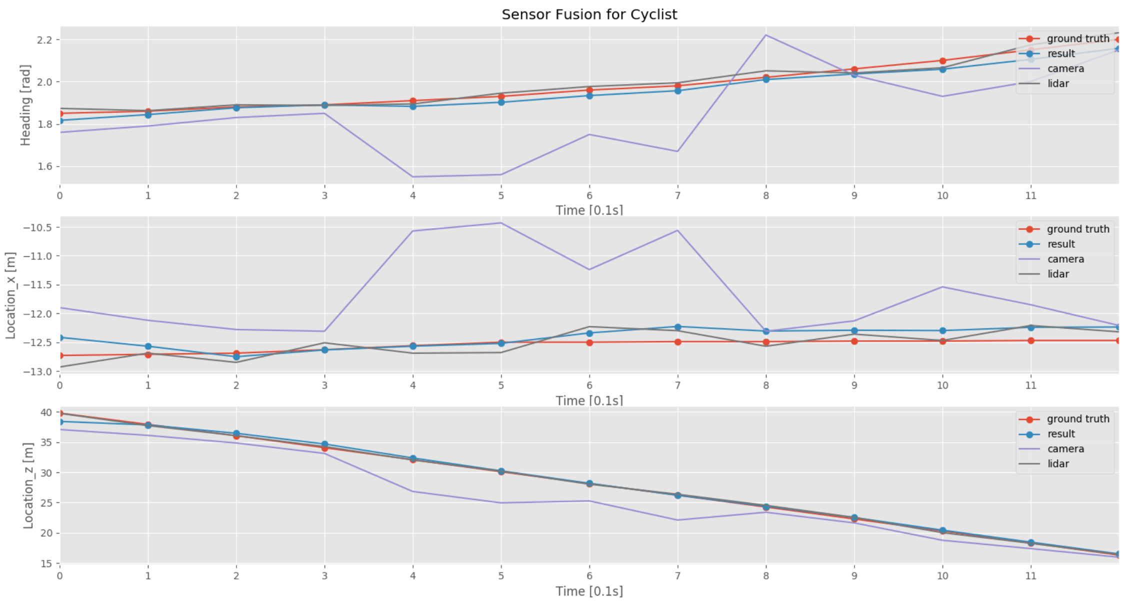


Figure 4.7: Sensor Fusion result for Cyclist.

4.5 Visualization of Detection Results

The visualization of 2D and 3D detection results are shown in the following figures. As can be seen from the figures, we achieve relatively accurate results on both cases. In Figure 4.9, we show some failed detection results from our method. The most common failure case is redundant bounding box detection on the same object. This is probably because the NMS described in Section 3.3.2 could not filter out all the irrelevant bounding boxes after RPN procedure.

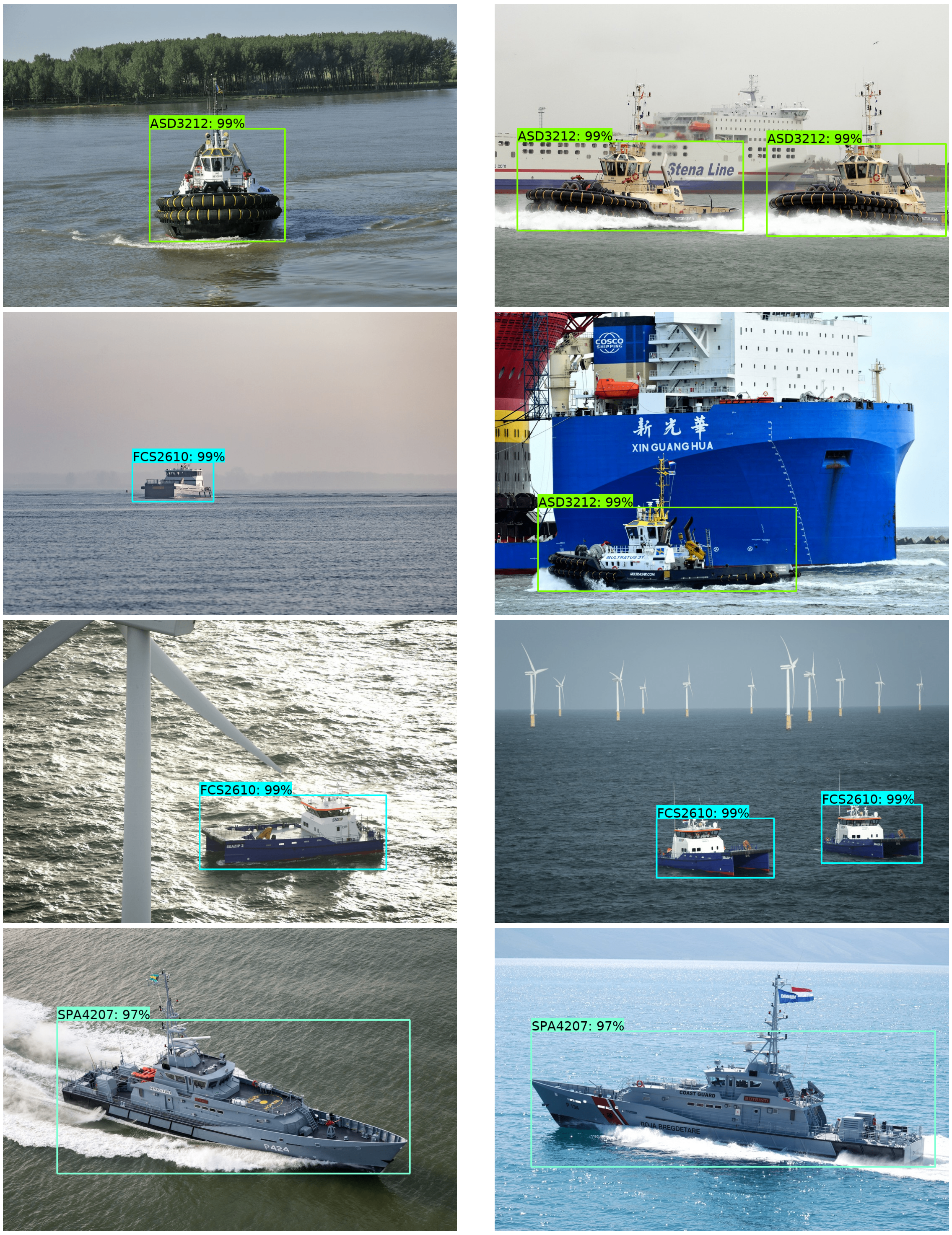


Figure 4.8: Selected successful examples of 2D ship detection results.

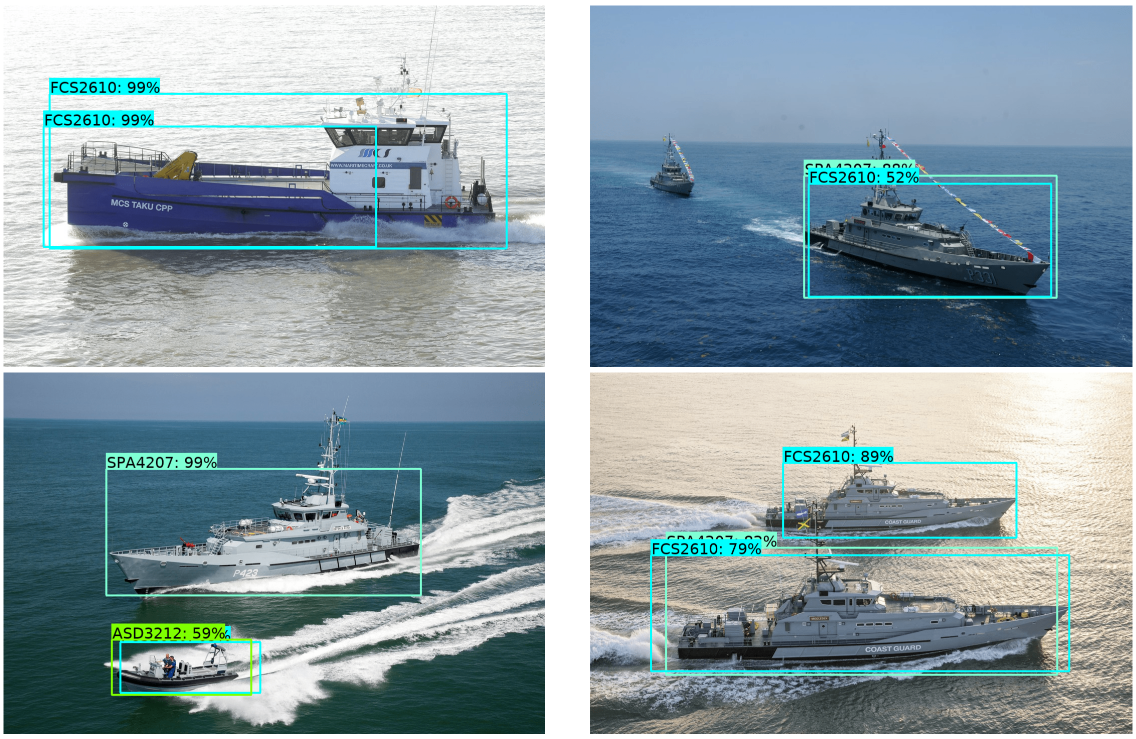


Figure 4.9: Selected examples of 2D ship detection contains failed detection results.

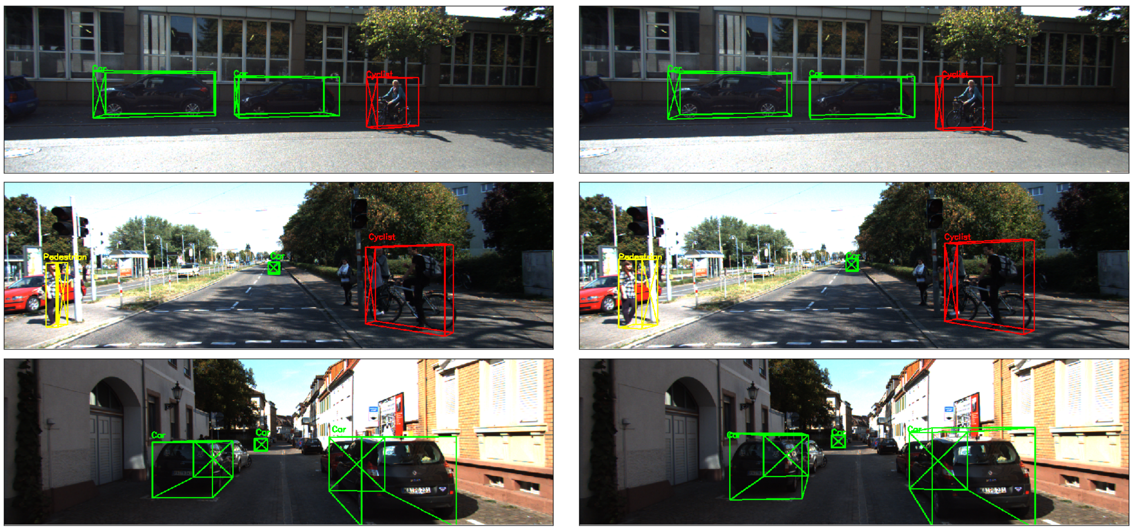


Figure 4.10: Selected examples of 3D bounding box estimation results on KITTI dataset. Left: ground truth 3D bounding box. Right: estimated 3D bounding box

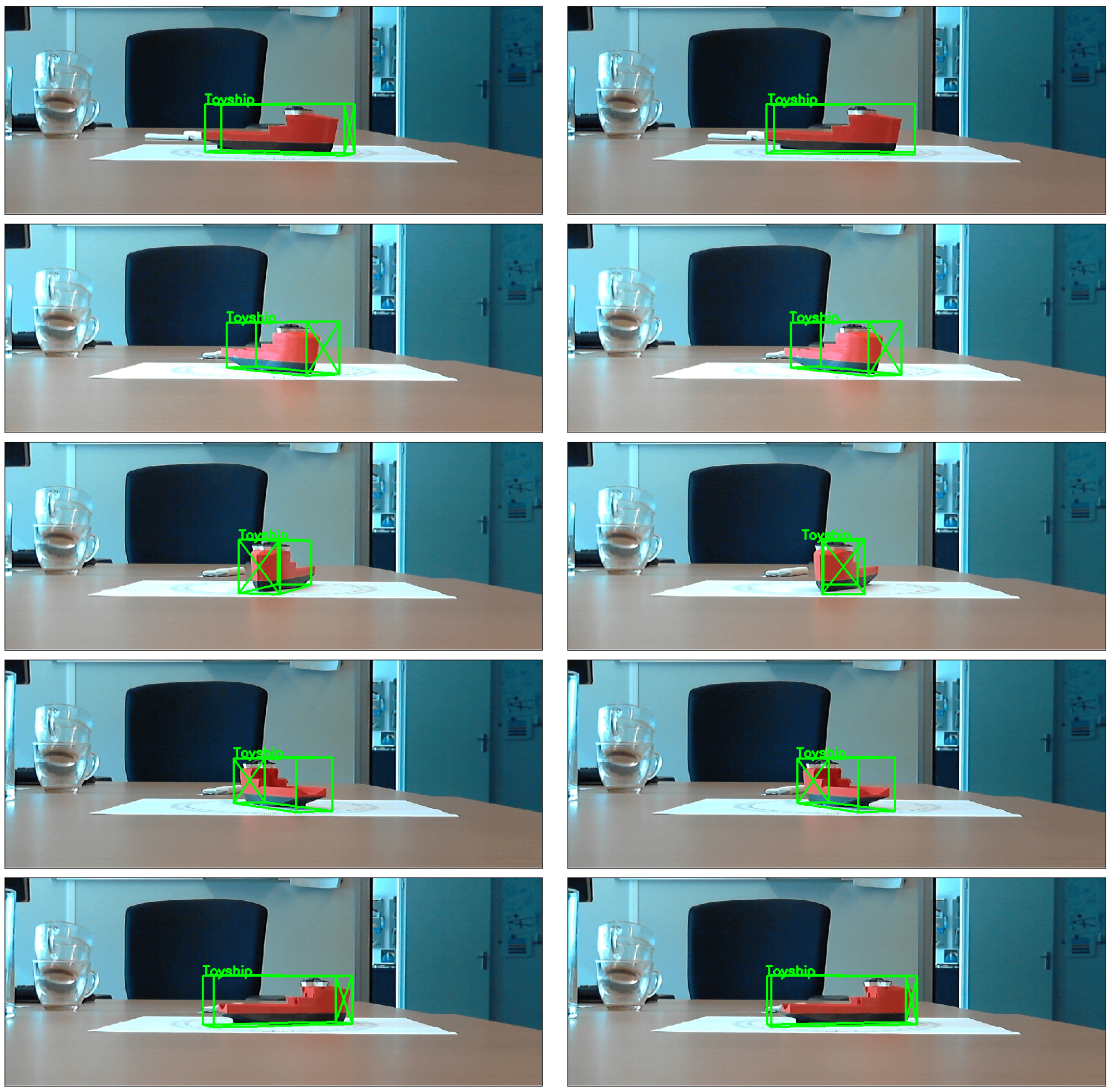


Figure 4.11: Selected examples of 3D bounding box estimation results on toy ship dataset. Left: ground truth 3D bounding box. Right: estimated 3D bounding box.

Chapter 5

Conclusions

In this chapter, the work presented in this thesis is summarized first. After that, the limitations of our proposed approaches will be described. And the last part will list the possible directions for future work.

5.1 Summary

In this project, we achieved promising detection results on both 2D and 3D scenes by a single RGB camera. Additionally, we created a small dataset on a toy ship to evaluate our ConvNet performance. The 2D and 3D ship detection both present accurate results. Furthermore, the ConvNet show good running efficiency which means it is promising to be applied to real autonomous ships in the future. The following paragraphs will show the major contributions of our work.

We first apply 2D object detection algorithm Faster R-CNN on a small ship dataset created by ourselves. Besides generating 9 anchors on each location on feature maps, 3 more anchors with smaller size are proposed additionally considering the real marine traffic condition. The result shows a mean average precision of 0.728 for all three types of ships. The resulting 2D detection algorithm need approximately 166ms to process one image on a modern graphic card.

Secondly, the main contribution of this project is successfully estimating 3D bounding box from a monocular camera. The original ConvNet are modified with Dropout and LeakyRelu. Also, random flip on images is added to make the ConvNet more robust. The new model can successfully predict accurate and stable orientation angle along with dimensions for different objects. It outputs accurate result on both KITTI and toy ship dataset. The orientation error for cars and toy ships are $2.7°$ and $11°$ respectively. The translations can also be solved by using camera matrices and some assumptions. Furthermore, this model needs 200ms time to process one image.

Finally, we propose a Kalman filter based sensor fusion technique to improve the detection result. The information from Lidar and camera are fused together and more accurate results can be obtained. By choosing which sensor information is more trustful, the angle error can reduce significantly from $11.45°$ to $1.51°$ as shown in Section 4.4.3.

5.2 Limitations

There are also some limitations that can be observed in our approach. The first disadvantage of our method is that we have two separate branches for 2D detection and 3D detection. These two parts are also trained independently. Normally in a deep learning based 3D object detection algorithm, single ConvNet which can predict 3D bounding box directly on images are constructed and trained. Our method takes more time for prediction and the 3D bounding box depends heavily on the 2D object detection precision. However, this modularity can be useful to subdivide the tasks and retrain specific parts of the network efficiently if new dataset is available.

The assumption that the 3D bounding box fits tightly in the estimated 2D bounding box does not always work. For example, if one ship is leaving the image boundaries and only part of the ship is visible, a problem with the given assumption occurs. Our approach can still predict the orientation, but it will fail at constructing the 3D bounding box. This is because for an object which is partly visible in the image, its ground truth 3D bounding box exceeds the image boundaries. In addition, the translation computation depends heavily on the estimated orientation and dimension values which results in relatively high estimation errors if any of the taken assumptions are violated.

5.3 Future Work

5.3.1 Dataset

Since deep learning is a data-driven approach. The dataset has a huge influence on the performance of the algorithm. Without a proper dataset, it is difficult to conduct further research on any perception tasks such as object detection and semantic segmentation. Thus, I recommend to first create a large image based dataset which records the real marine traffic scenes. The two papers published by KITTI dataset [62] and [69] would be nice references. In order to record such dataset, an autonomous shipping platform should be built at first and it should be equipped with several high-resolution video cameras, laser scanner, and other useful sensors such as GPS and IMU. The recording platform should travel on river or sea for days or weeks to store real marine traffic images and then all these images need to be labeled with full information. According to [62], all the labelling work is done manually and the information is obtained by 3D point clouds from the laser scanner. Therefore, such a dataset requires serious investment.

5.3.2 Detection Related

Various future application in detection related field can be applied once the dataset is prepared. Firstly, the base ConvNet in 2D detection can be modified to a modern network structure such as ResNet-152 [30] or ResNext [77] to increase the detection accuracy. Furthermore, RetinaNet [57] which is a more advanced 2D object detection algorithm can be applied to compare its performance with Faster R-CNN. Even instance segmentation approach such as Mask R-CNN [59] can be applied to conduct per pixel classification for different objects.

For 3D ship detection, one possible direction is to find a reasonable way to predict the translation since the three values for translation are below expectancy in our approach. Another option is to construct a sensory-fusion based network to combine Lidar images with camera images to generate more accurate and sufficient result [78].

Instead of camera image-based detection, some research can be conducted on other related fields such as object tracking and Lidar image-based detection. Object detection method can only identify and localize different objects in a scene. It does not combine time series data together. However, object tracking looks into different moving objects in real-time and can output continuous information.

5.3.3 Application in Ship Industry

In order to build an autonomous ship successfully, camera image-based detection method alone is not accurate to perceive the surrounding environment in marine traffic. It works poorly at night or at poor weather condition such as foggy or cloudy weather. Multiple sensors are often equipped onboard to detect and localize objects. Lidar could localize objects with better accuracy and it can work at night. However, lidar also performs poorly in bad weather conditions and its operating distance is only around 300 meters. Radar can achieve better detection results in cloudy weather and it has longer detection distance. Nevertheless, its performance on small objects is inadequate. A better way is to combine information from different sensors together in order to achieve fully autonomous vessels. In that direction, our proposed methodology certainly represent a step. There are also other interesting problems to investigate such as the communication time and time delay of different sensors.

Bibliography

[1] M. Schiaretti, L. Chen, and R. Negenborn. Survey on autonomous surface vessels: Part i-a new detailed definition of autonomy levels. *International Conference on Computational Logistics (ICCL)*, 2017. 1

[2] M. Schiaretti, L. Chen, and R. Negenborn. Survey on autonomous surface vessels: Part ii-categorization of 60 prototypes and future applications. *International Conference on Computational Logistics (ICCL)*, 2017. 1

[3] M. Blanke, M. Henriques, and J. Bang. A pre-analysis on autonomous ships. *Technical University of Denmark*, 2018. 1

[4] K. Kim, S. Hong, B. Choi, and E. Kim. Probabilistic ship detection and classification using deep learning. *Applied Sciences*, 2018. 1, 15

[5] S. Pendleton, H. Andersen, X. Du, X. Shen, Ma. Meghjani, Y. Eng, D. Rus, and M. Ang. Perception, planning, control, and coordination for autonomous vehicles. *Machines*, 2017. 1

[6] L. Tai and M. Liu. Deep-learning in mobile robotics-from perception to control systems: A survey on why and why not. *arXiv preprint arXiv:1612.07139*, 2016. 1

[7] D. Lowe. Object recognition from local scale-invariant features. *IEEE International Conference on Computer Vision (ICCV)*, 1999. 2

[8] P. Viola and M.J. Jones. Rapid object detection using a boosted cascade of simple features. *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2001. 2

[9] P. Viola and M.J. Jones. Robust real-time face detection. *IEEE International Conference on Computer Vision (ICCV)*, 2004. 2

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, A. Huang, Z.and Karpathy, M. Khosla, A.and Bernstein, et al. Imagenet large scale visual recognition challenge. *IEEE International Conference on Computer Vision (ICCV)*, 2015. 2, 11, 38

[11] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. *@inproceedingsioffe2017batch, title=Batch renormalization: Towards reducing minibatch dependence in batch-normalized models, author=Ioffe, Sergey, booktitle=Advances in Neural Information Processing Systems, pages=1945–1953, year=2017* , 2012. 2, 8, 11

[12] S. Ren, K. He, R. Girshick, and J. Sun. Faster R-CNN: Towards real-time object detection with region proposal networks. *Advances in Neural Information Processing Systems (NIPS)*, 2015. 2, 14, 15, 24

[13] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C. Fu, and A.C. Berg. Ssd: Single shot multibox detector. *European Conference on Computer Vision (ECCV)*, 2016. 2, 14, 15, 24

[14] X. Chen, K. Kundu, Y. Zhu, A. Berneshawi, H. Ma, S. Fidler, and R. Urtasun. 3d object proposals for accurate object class detection. *Advances in Neural Information Processing Systems (NIPS)*, 2015. 2, 14, 40

[15] Y. Xiang, W. Choi, Y. Lin, and S. Savarese. Subcategory-aware convolutional neural networks for object proposals and detection. *IEEE International Conference on Applications of Computer Vision (WACV)*, 2017. 2, 15, 16, 40

[16] F. Chabot, M. Chaouch, J. Rabarisoa, C. Teulière, and T. Chateau. Deep manta: A coarse-to-fine many-task network for joint 2d and 3d vehicle analysis from monocular image. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 15, 16

[17] H. Cho, Y. Seo, B. Kumar, and R. Rajkumar. A multi-sensor fusion system for moving object detection and tracking in urban driving environments. *IEEE International Conference on Robotics and Automation (ICRA)*, 2014. 2

[18] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014. 2, 14, 20

[19] R. Girshick. Fast R-CNN. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 2, 14, 17, 19

[20] A. Mousavian, D. Anguelov, J. Flynn, and J. Košecká. 3d bounding box estimation using deep learning and geometry. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 2, 15, 25, 38, 40

[21] D.H. Hubel and T.N. Wiesel. Receptive fields, binocular interaction and functional architecture in the cat's visual cortex. *The Journal of physiology*, 1962. 3

[22] A. Karpathy. Stanford university: Convolutional neural networks for visual recognition. 2017. 3, 4, 6, 7, 8

[23] V. Nair and G.E. Hinton. Rectified linear units improve restricted boltzmann machines. *IEEE International Conference on Machine Learning (ICML)*, 2010. 3, 8

[24] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015. 3, 8, 38

[25] H. Robbins and S. Monro. A stochastic approximation method. *Herbert Robbins Selected Papers*, 1985. 4

[26] D.P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 4

[27] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, pages 2278–2324, 1998. 5, 6

[28] I. Goodfellow, Y. Bengio, and A. Courville. *Deep learning*, volume 1. MIT press Cambridge, 2016. 5

[29] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 6, 9, 11, 17

[30] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 6, 9, 11, 15, 17, 48

[31] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014. 8

[32] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. *International Conference on Artificial Intelligence and Statistics (AISTATS)*, 2010. 8, 9

[33] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 1994. 8, 9

[34] D.R. Jean. $https://www.quora.com/how-is-a-convolutional-neural-network-able-to-learn-invariant-features$. 9

[35] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015. 9

[36] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research (JMLR)*, 2014. 9, 10, 38

[37] S. Ioffe. Batch renormalization: Towards reducing minibatch dependence in batch-normalized models. 2017. 9

[38] Y. Wu and K. He. Group normalization. *arXiv preprint arXiv:1803.08494*, 2018. 9

[39] A. Ng. Feature selection, l 1 vs. l 2 regularization, and rotational invariance. *IEEE International Conference on Machine Learning (ICML)*, 2004. 11

[40] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 11

[41] G. Huang, Z. Liu, L. Van Der Maaten, and K. Weinberger. Densely connected convolutional networks. *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 11

[42] J. Justin. $https://github.com/jcjohnson/cnn-benchmarksresnet-cvpr$. 2017. 11

[43] A. Das, S. Roy, and U. Bhattacharya. Document image classification with intra-domain transfer learning and stacked generalization of deep convolutional neural networks. *arXiv preprint arXiv:1801.09321*, 2018. 12

[44] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013. 11

[45] K. He and J. Sun. Convolutional neural networks at constrained time cost. *IEEE International Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015. 12

[46] R. K. Srivastava, K. Greff, and J. Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 12

[47] K. He, X. Zhang, S. Ren, and J. Sun. Identity mappings in deep residual networks. *European conference on computer vision (ECCV)*, 2016. 12

[48] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *The 22nd ACM international conference on Multimedia*, 2014. 13

[49] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer. Automatic differentiation in pytorch. 2017. 13

[50] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. *Conference on Operating Systems Design and Implementation (OSDI)*, 2016. 13

[51] T. Chen, M. Li, Y. Li, M. Lin, N. Wang, M. Wang, T. Xiao, B. Xu, C. Zhang, and Z. Zhang. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *arXiv preprint arXiv:1512.01274*, 2015. 13

[52] PaddlePaddle. *https : //github.com/paddlepaddle/paddle*. 2016. 13

[53] F. Chollet et al. Keras, 2015. 13

[54] J. Uijlings, K. Van De Sande, T. Gevers, and A. Smeulders. Selective search for object recognition. *IEEE International Conference on Computer Vision (ICCV)*, 2013. 14

[55] C. Zitnick and P. Dollár. Edge boxes: Locating object proposals from edges. *European Conference on Computer Vision (ECCV)*, 2014. 14

[56] M. Everingham, L. Van Gool, C.K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *IEEE International Conference on Computer Vision (ICCV)*, 2010. 14, 37

[57] T. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*, 2017. 14, 48

[58] T. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S.J. Belongie. Feature pyramid networks for object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 14

[59] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask R-CNN. *IEEE International Conference on Computer Vision (ICCV)*, 2017. 14, 48

[60] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 14

[61] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun. Monocular 3d object detection for autonomous driving. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016. 14, 16, 40

[62] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 15, 23, 48

[63] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 15

[64] A. Ghahremani, Y. Bondarau, et al. Self-learning framework with temporal filtering for robust maritime vessel detection. 2017. 16

[65] T. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick. Microsoft coco: Common objects in context. *European Conference on Computer Vision (ECCV)*. 16, 37

[66] M. Vahid. $https : //www.quora.com/how-does-the-region-proposal-network-rpn-in-faster-r-cnn-work$. 19

[67] P.F. Felzenszwalb, R. Girshick, D. McAllester, and D. Ramanan. Object detection with discriminatively trained part-based models. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2010. 20

[68] R. I. Hartley and A. Zisserman. *Multiple View Geometry in Computer Vision*. Cambridge University Press, ISBN: 0521540518, second edition, 2004. 22

[69] A. Geiger, P. Lenz, C. Stiller, and R. Urtasun. Vision meets robotics: The kitti dataset. *International Journal of Robotics Research (IJRR)*, 2013. 22, 23, 48

[70] W. Elmenreich. Sensor fusion in time-triggered systems. 2002. 29

[71] R.E. Kalman. A new approach to linear filtering and prediction problems. *Journal of Basic Engineering*, 1960. 29

[72] R.E. Kalman and R.S. Bucy. New results in linear filtering and prediction theory. *Journal of Basic Engineering*, 1961. 29

[73] C. Tarin, H. Brugger, R. Moscardo, B. Tibken, and EP Hofer. Low level sensor fusion for autonomous mobile robot navigation. *Instrumentation and Measurement Technology Conference (IMTC)*, 1999. 29

[74] labelimg. *https : //github.com/tzutalin/labelimg*. 33

[75] A. Geiger, P. Lenz, and R. Urtasun. Are we ready for autonomous driving? the kitti vision benchmark suite. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012. 33

[76] KITTI raw dataset. *http : //www.cvlibs.net/datasets/kitti/raw_data.php*. 2012. 34

[77] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 48

[78] X. Chen, H. Ma, J. Wan, B. Li, and T. Xia. Multi-view 3d object detection network for autonomous driving. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017. 48