Instance Segmentation and 3D Multi-Object Tracking for Autonomous Driving

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

M.Sc. Nuri Hamed Muhammad Benbarka

aus Newport, Wales

Tübingen

2022

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	09.03.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter:	Prof. Dr. Andreas Zell
2. Berichterstatter:	Prof. Dr. Andreas Schilling

Abstract

Autonomous driving promises to change the way we live. It could save lives, provide mobility, reduce wasted time driving, and enable new ways to design our cities. One crucial component in an autonomous driving system is perception, understanding the environment around the car to take proper driving commands. This dissertation focuses on two perception tasks: instance segmentation and 3D multi-object tracking (MOT).

In instance segmentation, we discuss different mask representations and propose representing the mask's boundary as Fourier series. We show that this implicit representation is compact and fast and gives the highest mAP for a small number of parameters on the dataset MS COCO. Furthermore, during our work on instance segmentation, we found that the Fourier series is linked with the emerging field of implicit neural representations (INR). We show that the general form of the Fourier series is a Fourier mapped perceptron with integer frequencies. As a result, we know that one perceptron is enough to represent any signal if the Fourier mapping matrix has enough frequencies. Furthermore, we used INR to represent masks in instance segmentation and got results better than the dominant grid mask representation.

In 3D MOT, we focus on tracklet management systems, classifying them into count-based and confidence-based systems. We found that the score update functions used previously for confidence-based systems are not optimal. Therefore, we propose better score update functions that give better score estimates. In addition, we used the same technique for the late fusion of object detectors. Finally, we tested our algorithm on the NuScenes and Waymo datasets, giving a consistent AMOTA boost.

Kurzfassung

Das autonome Fahren wird unser Leben verändern. Es könnte Leben retten, für mehr Mobilität zu sorgen, die Zeit, die wir mit dem Autofahren verschwenden, zu verkürzen und neue Wege für die Gestaltung unserer Städte zu eröffnen. Eine entscheidende Komponente eines autonomen Fahrsystems ist die Perzeption, d. h. das Verstehen der Umwelt um das Fahrzeug herum, um angemessene Fahrbefehle zu erteilen. Diese Dissertation konzentriert sich auf zwei Aufgaben der Perzeption: Instanzsegmentierung und 3D-Multi-Objekt-Tracking (MOT).

Bei der Instanzsegmentierung diskutieren wir verschiedene Maskendarstellungen und schlagen vor, die Grenzen der Maske als Fourier-Reihen darzustellen. Wir zeigen, dass diese implizite Darstellung kompakt und schnell ist und die höchste mAP für eine kleine Anzahl von Parametern auf dem Datensatz MS COCO liefert. Darüber hinaus haben wir während unserer Arbeit an der Instanzsegmentierung herausgefunden, dass die Fourier-Reihe mit dem aufkommenden Feld der impliziten neuronalen Repräsentationen (INR) verbunden ist. Wir zeigen, dass die allgemeine Form der Fourier-Reihe ein Fourier-abgebildetes Perzeptron mit ganzzahligen Frequenzen ist. Daraus ergibt sich, dass ein Perzeptron ausreicht, um ein beliebiges Signal darzustellen, wenn die Fourier-Mapping-Matrix genügend Frequenzen aufweist. Außerdem haben wir INR zur Darstellung von Masken bei der Instanzsegmentierung verwendet und dabei bessere Ergebnisse erzielt als mit der dominanten Gittermasken-Darstellung.

In 3D MOT konzentrieren wir uns auf Tracklet-Management-Systeme, die wir in zählbasierte und vertrauensbasierte Systeme unterteilen. Wir haben festgestellt, dass die bisher für vertrauensbasierte Systeme verwendeten Funktionen zur Aktualisierung der Punktzahl nicht optimal sind. Daher schlagen wir bessere Funktionen zur Aktualisierung der Punktzahl vor, die eine bessere Schätzung der Punktzahl ermöglichen. Darüber hinaus haben wir die gleiche Technik für die späte Fusion von Objektdetektoren verwendet. Schließlich haben wir unseren Algorithmus an den NuScenes- und Waymo-Datensätzen getestet, was zu einem konsistenten verbesserung von AMOTA führte.

Acknowledgments

I glorify and thank Allah for all His blessings and for sending us his prophet Mohammad; peace be upon him, who is my role model in this life. I thank my grandparents who passed away and could not see this moment. I thank them for raising me, and I ask Allah to have mercy on them. I also thank my Father and Mother for their continuous support. In addition, all my family, including my uncles and aunts, were like my big brothers and sisters. I also thank my wife for her love, support, and patience with me in hard times and for raising our lovely two sons.

Furthermore, I thank Prof. Andreas Zell for all his advice and guidance during my Ph.D. and his help during demanding times; I appreciate it. Finally, I would like to thank my colleagues, who joined me in the Ph.D. journey with their discussion and feedback and sharing of emotions. Particularly I thank my coauthors, Hamd, Timon, Jona, and Anwar, without whom I could not have finished this thesis.

Contents

Chapter 1

Introduction

1.1 Motivation

Autonomous driving is making vehicles drive without human intervention. Autonomous driving could transfer the way we live and has many advantages. The most important advantage is saving lives by reducing road accidents, as they are the leading cause of death for people between the age of 5 and 29, and the 8-th cause overall as shown in figure 1.1. And the main reasons for those accidents are speeding, drowsiness, distraction, intoxication, or not obeying safety rules (WHO, 2018). Another advantage is that it could provide mobility to people who cannot drive, such as the elderly, people with special needs, or teenagers. It could also enable new ways of public transportation such as carpooling and car-sharing, which would eventually reduce the number of cars and reduce pollution (Ratti, 2021). However, on the other hand, it could lead to a rise in unemployment, eventually leading to personal and social problems such as poverty, erosion of self-esteem, and family tensions (Alison McClelland, 1998). Furthermore, there is a possibility of data privacy problems, where companies can use their car fleet to collect real-time data of entire cities (VPRO, 2019).

For self-driving cars to run on public roads, they need to reach a specific driving performance. Otherwise, they will do more harm than good. Therefore, we compare it to the driving performance of current users of the public roads, which are humans. Although the main contributor to road accidents is poor human driving behavior, their overall driving performance is good. Statistically, in the USA, one person is killed in an accident for every 90 million miles (NHTSA, 2019), which is challenging for a machine to achieve.

In addition, humans are excellent at using their accumulated knowledge and can use it to learn how to behave in unseen environments efficiently. On the other hand, machines perform poorly in environments they never saw during the training phase; this drop in performance when given unseen data is called the generalization problem. For example, suppose we train a self-driving car on data collected in Europe. In that case, we can not let it run in Africa without further training because of the different general appearance, weather, road conditions, etc.

Another aspect that humans are good at is negotiating with other traffic participants. Humans can predict other people's intent by tiny hints, and if their prediction is wrong,

Rank	Cause	% of total deaths
	All Causes	
1	Ischaemic heart disease	16.6
2	Stroke	10.2
3	Chronic obstructive pulmonary disease	5.4
4	Lower respiratory infections	5.2
5	Alzheimer's disease and other dementias	3.5
6	Trachea, bronchus, lung cancers	3.0
7	Diabetes mellitus	2.8
8	Road traffic injuries	2.5
9	Diarrhoeal diseases	2.4
10	Tuberculosis	2.3

Figure 1.1: The estimate of the leading causes of death of all ages in 2016 by the World health organization (WHO, 2018).

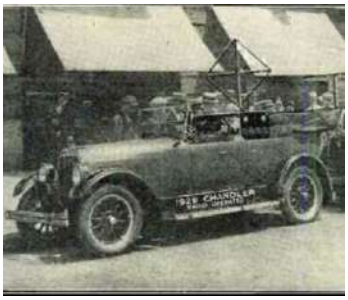
they can act to save the situation. However, this kind of reasoning is still challenging for machines. Two tasks that the machines would do and humans cannot do are full attention on driving and total obedience to traffic rules.

1.2 History of Autonomous driving

The history of driverless cars started some time after the invention of the car by Benz in 1886 (MBG, 2022) as shown in figure 1.2a. The first driverless car was made in 1925 and was called the ”American Wonder.” It did not have a driver in the car; however, it was remote-controlled by an operator sitting in a car following it as shown in figure 1.2b. A radio company built the car as a marketing strategy to show the capabilities of their radio control system (Engelking, 2017). Then the idea of self-driving was geared toward intelligent infrastructure rather than smart cars. For example, one idea was to bury cables under roads to control cars. This idea was first exhibited in the World's fair in 1939 and realized in 1960 by Criterion (KRA, 2014). Unfortunately, the idea of making smart infrastructure did not continue because it was expensive. Later, scientists decided to go in the other direction and develop smart cars instead of intelligent infrastructure.

In 1986, there were two parallel efforts to make an autonomous vehicle with no external control. One effort was a project by the NavLab at Carnegie mellon university (Jochem *et al.*, 1995) as shown in figure 1.2d, and the other effort was the two projects, VaMoRs (Dickmanns *et al.*, 1994) and VAMP (Maurer *et al.*, 1996), by Dickmann's group at Bundeswehr Universität Munich. The Navlab project resulted in the ”No Hands Across America” trip in 1995, where they could make the car drive semi-autonomously

(a) Benz patent car



(b) American wonder

(c) VaMoRs

(d) NavLab

(e) VAMP

(f) Stanley

Figure 1.2: A historic timeline of autonomous vehicles: (a) Benz patent motor car (MBG, 2022), the invention of the car in 1886 , (b) the first remote controlled car in 1925 Engelking (2017), (c) VaMoRs (Dickmanns *et al.*, 1994), (d) NavLab Jochem *et al.* (1995) and (e) VAMP Maurer *et al.* (1996) parallel first attempts to a self contained autonomous vehicle in 1986. (f) Stanley (Thrun *et al.*, 2006) the car that won the second DARPA grand challenge in 2005.

(only lateral control) for 2850 miles and 98% of the time. The VaMoRs and VAMP project resulted in a 1678km trip from Munich to Odense, where they could make the car drive autonomously for 95% of the time. The van used in the VaMoRs project is shown in figure 1.2c.

The previous projects showed the potential of autonomous driving; as a result, the US Defense Advanced Research Projects Agency (DARPA) organized three autonomous driving challenges in 2004, 2005, and 2007 (Behringer *et al.*, 2004; Tuttle *et al.*, 2007). In the first DARPA grand challenge, the participants should make a car drive fully autonomous along a 240 km route in the Mojave desert. The organizer gave 2935 Global Positional System (GPS) points to guide the cars through the desert. Unfortunately, none of the teams could finish the route. The following year DARPA organized the second challenge in the same desert but with a different route. This time five teams could finish the challenge, where Stanley (Thrun *et al.*, 2006) (shown in Figure 1.2f) completed the route in the shortest time and won the challenge. Finally, in the DARPA urban challenge, the participants had finished a 96 km route; however, this time in an urban scenario. The participants had to obey traffic rules, negotiate, avoid obstacles, and merge into traffic.

Figure 1.3: The autonomous driving platform Annieway which was used to record the KITTI dataset. This car contains multiple cameras, a LIDAR, a GPS, and an IMU (Geiger *et al.*, 2012).

Six teams finished this challenge, which showed that it is possible to achieve autonomous driving in a controlled environment. One of the enablers of these achievements was the development of GPS and IMUs, which allowed vehicles to have a cm range localization accuracy. In addition, the development of LIght Detection And Ranging (LIDAR) sensors gave accurate 3D reconstructions of the vehicle surroundings (Thrun *et al.*, 2006).

In 2012, the deep learning revolution came with the availability of large datasets and high computing power, mainly by GPUs. At that time, the Karlsruhe Institute of Technology and Toyota Technological Institute (KITTI) dataset benchmarked many of the autonomous driving tasks (Geiger *et al.*, 2012), which was recorded with Annieway shown above 1.3. This benchmarking made it easy for small research groups to focus on small subtasks and enter the development race. At the same time, there was a massive investment in autonomous driving. Either by startups like Waymo, NuTonomy, Zoox, etc., who want to solve autonomous driving directly, or by car companies like Daimler, Toyota, Ford, etc., who want to introduce the technology gradually. In this deep learning era, three main approaches appeared to solve autonomous driving, which we will discuss in the following few sections. Before that, we will discuss the vehicle platform itself.

1.3 The autonomous vehicle platform

An autonomous vehicle is a regular vehicle with additional sensors to perceive the surroundings and onboard computing to run the algorithms. We typically equip the vehicle with cameras, LIght Detection And Ranging (LIDAR) sensors, RAdio Detection And Ranging (RADAR) sensors, SOund Detection And Ranging (SONAR) sensors, inertial

measurement units (IMU), GPS, and wheel odometry. Annieway is an example of an autonomous vehicle which is equipped with multiple cameras, A LIDAR, a GPS, and an IMU, as shown in figure 1.3. Cameras are sensors used to capture the reflected light from the scene. Cameras are relatively inexpensive and provide rich information about the environment. Because of their advantages, almost all companies equip their vehicles with cameras.

LIDARs, RADARs, SONARs are range sensors. They are used to find the distance of objects in the scene. They do this by sending waves to the environment and then measuring their reflection, and from the wave's time of travel and its speed, we can infer the object's distance. LIDARs use light waves, and they are the most accurate range sensors; however, they are bulky, expensive, and have problems when used in foggy and rainy weather. Therefore, the industry has made a considerable effort to reduce its cost and size, encouraging adoption. RADARs use electromagnetic waves, and they have a more extended range than LIDARs, are cheap, and can be used to infer the speed of objects. On the other hand, we can not recognize the shape of the surrounding objects because they are not accurate enough. SONAR use sound waves and are the smallest and cheapest. However, their update speed is slow, so they are mainly used in low-speed scenarios such as parking.

GPS is used to find the vehicle's absolute position on earth for localization. The GPS reads time-stamped satellite signals. Using the time-stamps, it could calculate the distance from each visible satellite, then using triangulation, it could find its position relative to the visible satellites. A typical GPS accuracy is 5 meters, so GPS alone can not be used for localization as this distance is larger than the lane width. IMUs are used alongside the GPS for localization. These sensors measure the vehicle's linear acceleration and rotational velocity; however, they are very noisy sensors, and should be used with other sensors. The sensors we chose to equip the vehicle determine the algorithms we can use to solve the driving task and determine its performance's upper bound.

1.4 Approaches to solving autonomous-driving

Solving autonomous driving can be seen as finding a mapping function between the high-dimensional sensor inputs and low-dimensional driving commands. There are three main ways to solve the driving task, as shown in figure 1.4: the classical modular pipeline approach, end-to-end learning, and direct perception.

In the classical modular pipeline approach, we divide the task into sub-tasks and solve each independently. These sub-tasks include: perception, which is understanding the environment around the car to take proper driving commands; perception tasks include object detection, tracking, instance segmentation, or optical flow. Localization is locating the vehicle itself on the map. Path-planning is finding a sequence of good moves that would make the vehicle reach the destination in an optimized way, for example, as fast and safe as possible. Finally, the control system gives the vehicle low-level com-

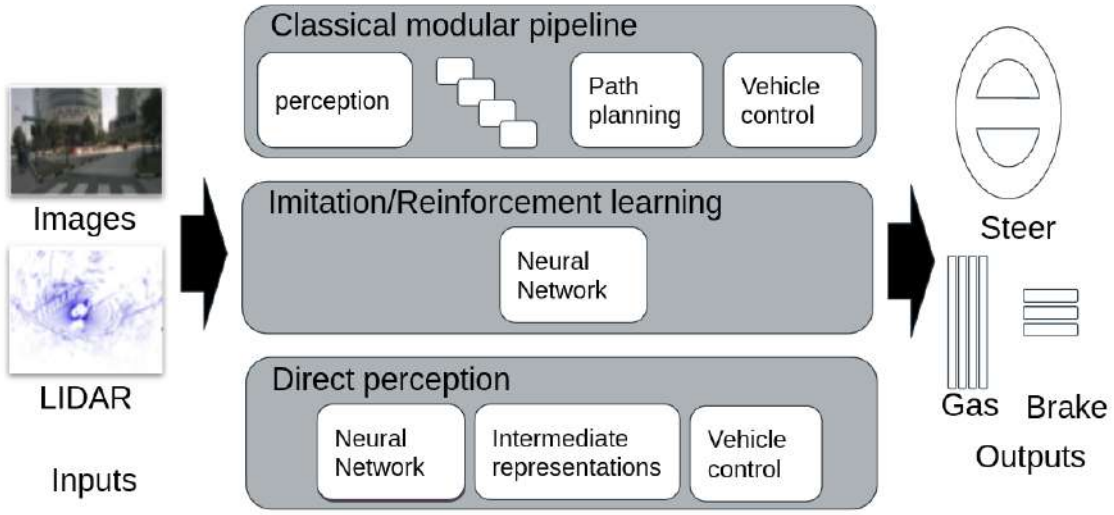


Figure 1.4: The three different approaches to solve autonomous driving.

mands to ensure the high-level plan is accomplished. Today, the modular pipeline is still used in many autonomous driving systems because it allows the developer to find and debug problems easily compared to the black box neural networks. In addition, there are many reliable and robust solutions to some sub-tasks, such as path planning and map localization. However, many works showed that one could get a performance boost if the network learns multiple tasks jointly. Nevertheless, the perception sub-task is the system's bottleneck, and it is far from being solved, so it is the focus of this thesis and the research community's focus in the past couple of years.

The second approach to self-driving is end-to-end learning. Here, the mapping function between the inputs and outputs is a neural network trained using imitation or reinforcement learning. In imitation learning, the network learns to imitate human drivers by providing cheap human driving data. The challenge in imitation learning is how to make the vehicle learn what to do in situations it did not witness. In reinforcement learning, we optimize the network for a predefined reward function representing the ideal driving behavior. However, finding such a reward function is not straightforward. In addition, there is an exploration phase when running reinforcement learning, where the car will try random actions to learn from its mistakes, which could be dangerous in the real physical world. Usually, we make this exploration phase in simulation; however, there is still the challenge of closing the gap between simulation and reality.

The third approach is direct perception, a hybrid of the first two methods. First, we use a neural network to get intermediate representations such as distance to the lanes, object detection, and optical flow. Then use these intermediate representations to control the vehicle. Controlling the vehicle can be either classical or learned-based. Zhou *et al.* (2019a) showed empirically that intermediate representations help in end-to-end learning

tasks, so it looks like the solution will not be an end-to-end approach. In our work in the thesis on perception tasks, we can use these tasks as a separate module in the modular pipeline approach or the first step to get intermediate representations in the direct perception approach. Nevertheless, the perception sub-task is the system's bottleneck, and it is far from being solved, so it is the focus of this thesis and the research community's focus in the past couple of years.

1.5 Contribution & Outline

This dissertation focuses on two perception tasks used in autonomous driving: instance segmentation and 3D tracking. In instance segmentation, we focus on mask representations. Furthermore, in 3D tracking, we focus on tracklet management systems and late fusion in 3D detection. This work is mainly based on the following four peer-reviewed conference papers:

1. Riaz, Hamd Ul Moqeet*, Nuri Benbarka*, and Andreas Zell. "FourierNet: Compact mask representation for instance segmentation using differentiable shape decoders." International Conference on Pattern Recognition (ICPR) 2021.

2. Benbarka, Nuri*, Timon Höfer*, Hamd Ul Moqeet Riaz and Andreas Zell. "Seeing implicit neural representations as Fourier series." IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2022.

3. Riaz, Hamd Ul Moqeet*, Nuri Benbarka*, Timon Höfer, and Andreas Zell. "FourierMask: Instance Segmentation using Fourier Mapping in Implicit Neural Networks." International Conference on Image Analysis and Processing (ICAIP) 2021.

4. Benbarka, Nuri, Jona Schröder, and Andreas Zell. "Score refinement for confidence-based 3D multi-object tracking." International Conference on Intelligent Robots and Systems (IROS) 2021.

In addition to the previous works, I contributed to the following works:

5. Höfer, Timon, Faranak Shamsafar, Nuri Benbarka and Andreas Zell. "Object detection and Autoencoder-based 6D pose estimation for highly cluttered Bin Picking". IEEE International Conference on Image Processing (ICIP) 2021.

6. Varga, Leon Amadeus, Martin Messmer, Nuri Benbarka and Andreas Zell. "Wavelength-aware 2D Convolutions for Hyperspectral Imaging." IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2023.

However, these works (Höfer *et al.*, 2021; Varga *et al.*, 2023) are unrelated to the main topic, so we will not discuss them in this dissertation. We will structure the rest of the dissertation as follows:

Chapter 2 We introduce the instance segmentation task and discuss the main approaches to solving this task. Then we introduce the different mask representations and the advantages and disadvantages of each representation. In addition, we introduce the primary datasets used to benchmark this task and the metrics used to evaluate it.

Chapter 3 We propose a compact mask representation, the Fourier series of the mask boundary. We show that it gives excellent performance compared to other masks with few parameters. However, our method is limited to instances with single star-shaped masks, so we extend our work to use the Fourier series of level sets to represent the masks.

Chapter 4 We describe the theoretical foundation for the general Fourier series equation to use it to express level sets. In addition, we show the mathematical connection between the equation and implicit neural representations. This connection gave us an understanding of MLPs with Fourier features. For example, a single Fourier mapped perceptron can represent any signal if it has enough frequencies, and elements of the mapping matrix represent the frequencies of the reconstructed signal.

Chapter 5 We discuss our second mask representation, using both a Fourier series and implicit neural representation to represent the level set. We show that our representation is better than the widely used grid-based masks.

Chapter 6 We introduce the 3D multi-object tracking task and explain its pipeline. We also introduce the most popular 3D multi-object tracking datasets and the evaluation metrics of this task.

Chapter 7 We discuss the problem with the current count-based and confidence-based tracklet management systems. Furthermore, we propose score update functions to refine the tracklet scores in confidence-based systems. Furthermore, we introduce a late fusion algorithm that relies on the confidence of the tracklets. With this work, we won the NuScenes tracking challenge of 2021. +

Chapter 8 This chapter concludes our work and gives some future work.

Chapter 2

Instance segmentation

2.1 Introduction

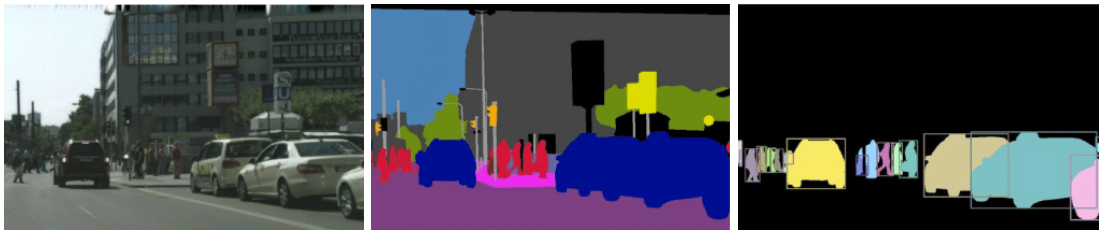
With the recent emergence of deep learning enabled by large datasets and high computational power, autonomous machines have become a practical option in many decision-making processes. One example of these decision-making processes is autonomous driving, and the first and foremost task in autonomous driving is perceiving and understanding the scene before acting.

Instance segmentation is one of the object recognition techniques which we use for scene understanding (Janai *et al.*, 2017). It is classifying each pixel of an image by a predefined class (or as known by semantic segmentation) and, at the same time, distinguishing different instance occurrences. Figure 2.1 shows the difference between instance and semantic segmentation. Instance segmentation deals with countable objects only, while semantic segmentation deals with both. This chapter discusses the main approaches to solve instance segmentation, and it introduces the different mask representations and the advantages and disadvantages of each representation. In addition, it introduces the datasets and metrics used to evaluate the task.

2.2 Convolutional Neural Networks (CNNs)

In recent years, most methods have employed CNNs for instance-segmentation. A CNN is an artificial neural network (ANN) that we typically use to analyze images. They use shared-weight convolution kernels that slide along input features and provide translation-equivariant activations known as feature maps. CNNs are inspired by biological creatures where the connectivity between neurons resembles the organization of the visual cortex. Cortical neurons react to stimulations only in a limited visual region known as the receptive field. The overlap of different neurons' receptive fields covers the entire visual field.

A simple CNN architecture is a series of layers sequentially transforming volumes of activations to another through differentiable functions. The three main layers to build CNN architectures are the convolutional layer, pooling layer, and activation layer. We



(a) Image (b) Semantic segmentation (c) Instance segmentation

Figure 2.1: For a given (a) image, we show ground truth for: (b) semantic segmentation (per-pixel class labels), (c) instance segmentation (per-object mask and class label). (Kirillov *et al.*, 2019)

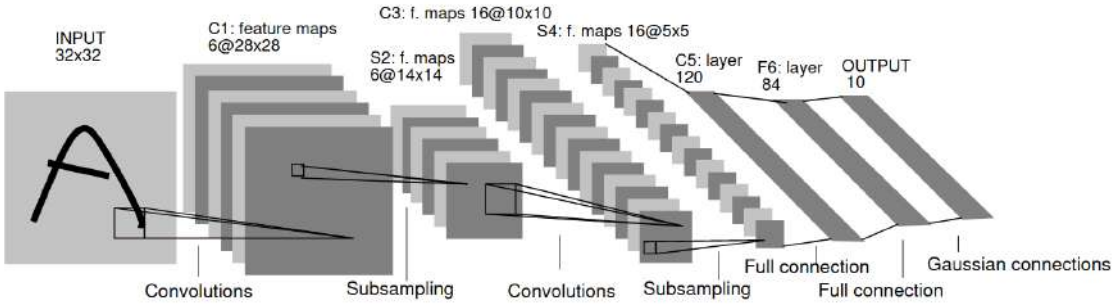


Figure 2.2: The architecture used in LeNet (LeCun *et al.*, 1998), a successful CNN for digit recognition. It has two convolution layers each followed by a pooling layer, then two fully connected layers and the final output.

convolve kernels to the previous feature map in the convolution layer to get a new feature map. We apply an elementwise activation function to the feature map in the activation layer, such as the Rectified Linear Unit (ReLU) function. We reduce the feature map size along the spatial dimensions (width, height) in the pooling layer by taking the maximum or average of local areas in the feature map. Finally, we pile these layers to form a complete CNN architecture.

Several CNN architectures became popular in the field, and one of those was LeNet by LeCun *et al.* (1998), they used it to read zip codes and digits. Figure 2.2 shows the architectures used in LeNet. After that, the most popular architectures were the ones that won the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) (Krizhevsky *et al.*, 2012) from 2012 to 2015, and we still use some of them now. Figure 2.3 shows the ImageNet's classification task's error over the years and the names of the CNN architectures that won or were runner-ups in the challenge.

In 2012, Krizhevsky *et al.* (2012) submitted AlexNet to the ILSVRC, which was the work that popularized CNNs in the current deep learning era. They significantly outperformed the runner-up in the top 5 error of 16% compared to a 26% error. The network

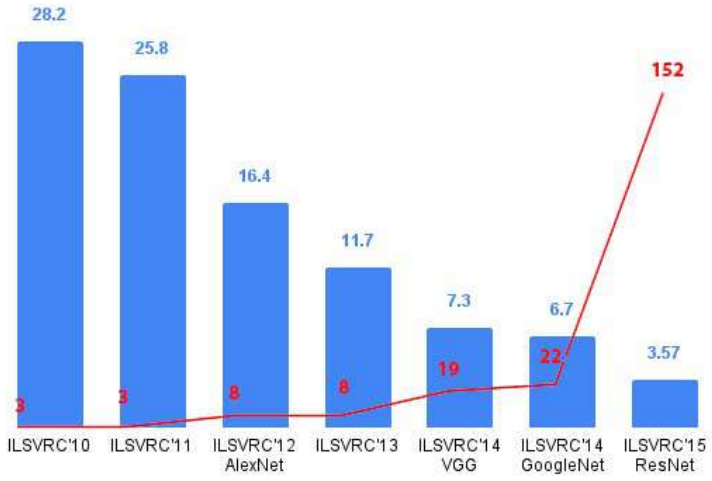


Figure 2.3: The ImageNet classification top-5 error from years 2012 to 2015 (He *et al.*, 2016). We can see the correlation between the reduction in error in blue and the number of layers of the networks in red.

had similar architecture to LeNet but was deeper and wider. The idea of using deep networks was novel at the time.

The ILSVRC winner of 2014 was a CNN from Google (Szegedy *et al.*, 2015) named GoogLeNet. Its main contribution was the Inception Module, which dramatically reduced the network's number of parameters from 60M to 4M. Additionally, their architecture uses Average Pooling instead of fully connected layers at the end of the CNN, removing many unneeded parameters. The runner-up in 2014 was VGGNet (Simonyan and Zisserman, 2014). They showed that the depth of the network is a critical component for good performance. Their final network contains 16 convolutional layers and features a homogeneous architecture that only performs 3x3 convolutions and 2x2 pooling.

He *et al.* (2016) developed the residual network, the winner of the ILSVRC of 2015. They introduced skip connections which solved the vanishing gradient problem and enabled the training of very deep networks (152 layers). ResNets are currently one of the default choices for using CNNs in practice. Unfortunately, there are no significant breakthroughs in network architectures for vision other than vision transformers (Liu *et al.*, 2021). However, some CNN architectures are worth mentioning, like inceptionV4 (Szegedy *et al.*, 2017), ResNeXt (Xie *et al.*, 2017), MobileNet (Howard *et al.*, 2017), and EfficientNet (Tan and Le, 2019). We use many of these CNN architectures as backbones for different recognition tasks, such as instance-segmentation in our case.

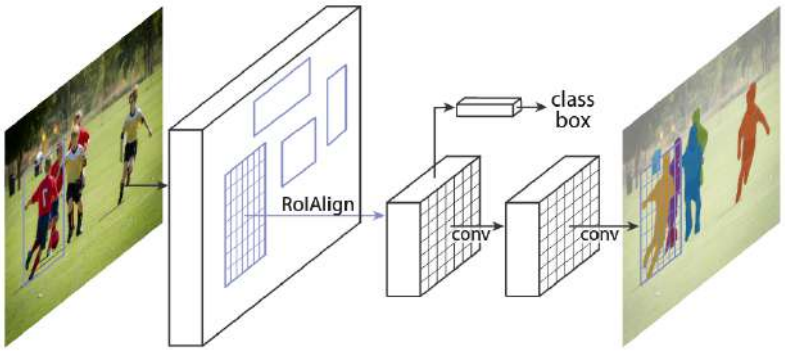


Figure 2.4: The Mask R-CNN (He *et al.*, 2017) framework for instance-segmentation. It contains an RPN that gives object proposals, then RoI-Align to reshape the proposal features to match the head size, and finally, three branches for the class, box parameters, and the mask.

2.3 Approaches to solve instance segmentation

We classify approaches to solve instance segmentation into two-stage and single-stage methods. Furthermore, we classify the two-stage methods into detect-then-segment and segment-then-cluster methods. The next paragraphs will explore these methods.

2.3.1 The detect-then-segment approach

These methods split the task into two subtasks, object detection, and segmentation. First, an object detector produces a bounding box, and then a segmentation network classifies the pixels within that box as foreground or background. The most well-known instance segmentation method is Mask R-CNN (He *et al.*, 2017), constructed on the object detector Faster R-CNN (Ren *et al.*, 2015). Faster R-CNN uses a CNN as a Region Proposal Network (RPN). This network classifies predefined anchor boxes distributed on the image to whether they contain an object or not. Then we reshape the predicted region proposals using a Region-of-Interest (RoI) pooling layer, which we use to classify the image within the proposed region and predict the offset values for the bounding boxes. Mask R-CNN added a mask branch to Faster R-CNN parallel to the bounding box and the classification branches as shown in figure 2.4 and used RoI-Align instead of RoI-Pooling.

Most detect-then-segment methods after Mask R-CNN were built on top of it. PANet (Liu *et al.*, 2018) improved the information flow from the backbone to the heads using bottom-up paths in the feature pyramid and adaptive feature pooling. In Mask Scoring R-CNN (Huang *et al.*, 2019), the network estimates the IoU of the predicted mask and uses it to refine the prediction scores. Hybrid Task Cascade (HTC) (Chen *et al.*, 2019)

introduced the cascade of masks by merging detection and segmentation features and achieved enhanced detections. ShapeMask (Kuo *et al.*, 2019) introduced class-dependent shape priors and used them as preliminary estimates to obtain the final detection.

CenterMask (Lee and Park, 2019) added a spatial attention-guided mask on top of the Fully Convolutional One-Stage object detector (FCOS)(Tian *et al.*, 2019). This spatial attention helped focus on critical pixels and diminished noise. After getting an initial mask, some detect-then-segment methods added a refinement step, such as PolyTransform (Liang *et al.*, 2019) and PointRend (Kirillov *et al.*, 2020). PolyTransform first converts the initial grid representation mask into a polygon representation mask. Then, they deform the polygon using a network to obtain the final crisp masks. PointRend predicts an initial low-resolution mask. Then, in uncertain positions in that mask, they take corresponding features to that position from the last feature map and pass them into an Multi-Layer Perceptron (MLP) to predict the occupancy at that position. This process is repeated recursively until a specific resolution, or there are no uncertain pixels. The MLP here implicitly represents the mask.

2.3.2 The segment-then-cluster approach

Segment-then-cluster methods first predict the class label of each pixel and then group them to form instance segmentation results through methods such as clustering and metric learning. These methods are generally less accurate than detect-then-segment methods. In addition, they need high computing power because of the dense prediction of the semantic segmentation.

De Brabandere *et al.* (2017) proposed a pixel-level discriminative loss function to deal with different instances. The loss function maps each pixel to a point in the feature space, making the distance between pixel's features belonging to the same instance close, while the distance between different instances is very far. Bai and Urtasun (2017) proposed to combine semantic segmentation with the traditional watershed algorithm in an end-to-end manner. First, they generate an energy map and then segment each instance in the map, which they can perform relatively quickly.

Kirillov *et al.* (2017) proposed to predict both semantic segmentation and instance boundaries. After predicting both, they perform semantic segmentation to get semantic information and use an edge detector to differentiate different instances. SSAP (Gao *et al.*, 2019) proposed to learn the probability that two pixels belong to the same instance. They do this by learning the affinity pyramid of pixel pairs. First, they use high-resolution images to learn the short-distance affinity and low-resolution images for the long-distance affinity. Then, they use short- and long-distance affinities to generate the multi-scale affinity pyramid, which they use with the semantic segmentation predictions to get instance predictions.

2.3.3 The single-stage approach

Single-stage instance segmentation methods predict instance masks in a single shot without using any proposed regions/bounding boxes as an intermediate step. These methods are usually more straightforward, faster than other approaches, but less accurate.

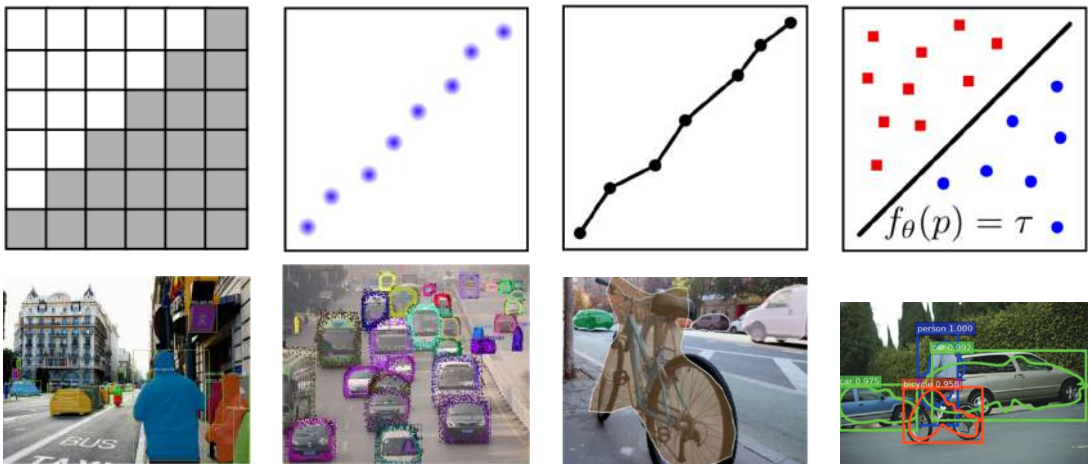
You Only Look At CoefficienTs (YOLACT) (Bolya *et al.*, 2019) generated prototype masks and simultaneously produced bounding boxes and combination coefficients. Then, they cropped the prototype masks with the bounding boxes and made a weighted sum of the cropped prototype masks using the combination coefficients to construct the final mask at real-time speeds. Likewise, Embedmask (Ying *et al.*, 2019) generated pixel embeddings that differentiate each instance in the image and simultaneously produced bounding boxes and proposal embeddings. Finally, they form the mask by comparing the proposal embedding with all pixel embeddings in the produced bounding box area.

ExtremeNet (Zhou *et al.*, 2019b) used keypoint detection to obtain the extreme points of an object. Then a rough mask was created by forming an octagon from the extreme points. Similarly, Dense RepPoints (Yang *et al.*, 2019) used keypoint detection to obtain numerous points. They get the mask using a meshing algorithm on the points or taking their concave hull.

Polarmask (Xie *et al.*, 2020) performed a dense regression of the distances from the mask center to points on the outer contour in polar coordinates. Additionally, since FCOS (Tian *et al.*, 2019) showed that the detections near object boundaries were generally inaccurate, they likewise adopted the concept of centerness, which gave greater importance to the detections near the center and enhanced the prediction quality. ESE-Seg (Xu *et al.*, 2019) has encoded the objects' contours using function approximations such as Chebychev polynomials and Fourier series. First, they predict the function parameters, or what they call the shape vector, and then they transform these parameters into contour points in the polar representation.

2.4 Mask representations

We can classify instance segmentation methods according to the mask representation. Mask representation is an important design choice because it could limit the system's performance if we do not choose it appropriately. For example, if the mask representation can only represent convex masks, all non-convex masks will only have convex approximations. There are four main mask representation methods as shown in figure 2.5: Grid, point, polygon, and implicit representations. The mask representation should be accurate, memory efficient, easy to train, and requires minimal post-possessing. Each of which has its advantages and disadvantages, and the following section we will talk about these representations.



a) Grid b) Point c) Polygon d) Implicit

Figure 2.5: An illustration of mask representations with examples (Mescheder *et al.*, 2019): a) The grid representation (Mask R-CNN (He *et al.*, 2017)). b)The point representation (Dense rep points (Yang *et al.*, 2019)). c)The polygon representation (PolarMask (Xie *et al.*, 2020)) .d)The implicit representation (ESE-Seg (Xu *et al.*, 2019)).

2.4.1 Grid representation

We represent the mask in the grid representation as a grid of pixels where we classify each pixel as foreground or background as shown in figure 2.5a). It is the most used representation due to its simplicity. However, it can only provide a discretized output approximation at low resolutions. Moreover, it needs too much memory for high-resolution masks, limiting some applications. Examples of grid representation methods are MaskR-CNN (He *et al.*, 2017) and YOLACT (Bolya *et al.*, 2019).

2.4.2 Point representation

We represent the mask in point representation as a set of pointsas shown in figure 2.5b). This representation can be accurate if we use a high number of points. However, it needs a post-processing step to get the actual mask. Dense rep points (Yang *et al.*, 2019) is the only method using the point representation, and they get the mask either using a meshing algorithm on the points or taking their concave hull.

2.4.3 Polygon representation

We represent the mask in the polygon representation as a set of connected points or a polygonas shown in figure 2.5c). The polygon representation can be compact, fast, and accurate. However, dealing with it in a deep learning framework is complex. Therefore, some methods made some restrictions to make it easy to deal with this representation.

For example, Polarmask (Xie *et al.*, 2020) allowed only star-shaped polygons, and Polytransform used it in the refinement step.

2.4.4 Implicit representation

We represent the mask in the implicit representation with a parameterized continuous function that maps the input domain of the mask (e.g., coordinates of a specific pixel in the image) to the value at that input location (e.g., foreground or background) as shown in figure 2.5d). This representation can be accurate if we use a proper function. However, it can be slow if the function takes some time to evaluate. Examples of methods that use implicit representation are ESE-Seg (Xu *et al.*, 2019), which represents the mask boundary as a Chebyshev polynomial, and the refinement step of PointRend (Kirillov *et al.*, 2020), representing the mask as multi layer perceptron.

2.5 Datasets

Datasets are crucial for deep learning algorithms. They contain the data needed for training along with the labels made by annotators. In addition, public datasets are used to benchmark algorithms to track the progress in solving the different tasks. Here we show two general-purpose and two autonomous driving focused instance segmentation datasets:

2.5.1 Pascal Voc

The Pascal VOC dataset is one of the first datasets for classification, object detection, action recognition, semantic segmentation, and instance segmentation. It was used in the PASCAL VOC Challenges between 2005 and 2012. It contains 11530 images with 20 classes. In addition, images have detailed annotation information. Therefore, it has been the essential benchmark before the large-scale application of the Microsoft Common Objects in Context dataset.

2.5.2 Microsoft Common Objects in Context (COCO) Dataset

The Microsoft COCO Dataset is a large-scale image dataset to recognize, segment, and caption images with diverse scenes. It has 82783 training images, 40504 validation images, and 80000 testing images annotated with 80 object classes. Due to its large scale and diversity, COCO got much interest since it first appeared. COCO became the standard benchmark in object detection and instance segmentation.

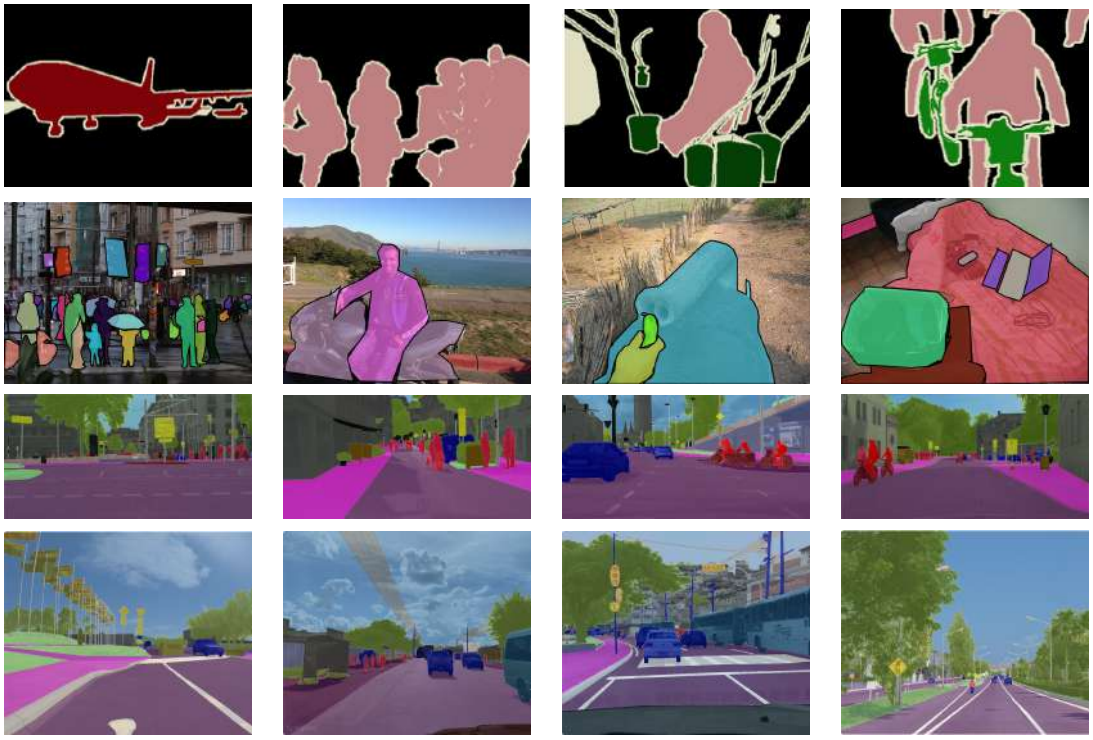


Figure 2.6: Samples of annotated images from the dataset (rows from top to bottom): a) Pascal Voc dataset, b) Microsoft Common Objects in Context Dataset, c) Cityscapes Dataset, d) The Mapillary Vistas Dataset

2.5.3 Cityscapes Dataset

The cityscapes dataset focuses on semantic understanding of the street scenes and provides semantic, instance-specific annotations. It has 30 object classes that have been grouped into eight categories relevant to urban scenes like vehicles, people, sky, and roads. It consists of about 5000 high resolution images with refined annotations and 20000 with coarse annotations. The images for the dataset were captured in fifty cities in Europe over several months during the daytime with good weather. The video frames were selected to have a diverse number of object classes, scenes, and backgrounds.

2.5.4 The Mapillary Vistas Dataset (MVD)

The MVD is another extensive street scene image dataset. It contains 25000 annotated images with 66 classes, and its annotations are dense and fine-grained. The dataset is five times bigger than Cityscapes for fine-annotation. It features images from across the world that have been captured during different weathers, seasons, and daytimes. In addition, different photographers have captured the dataset images using different devices like cell phones and cameras. The aim of developing the database is to further develop state-of-the-art research in understanding street scenes.

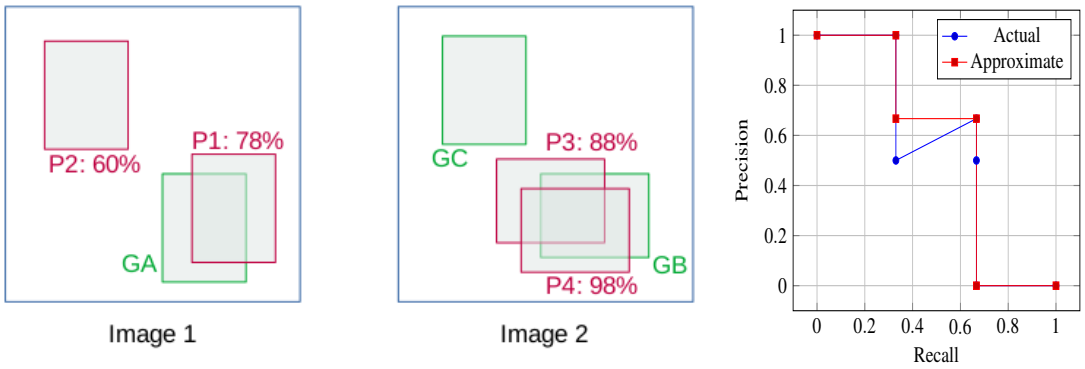
2.6 Evaluation

We evaluate to test the pros and cons of different algorithms. In evaluation, the algorithms are usually tested on the same dataset and the same hardware so that the comparison will be as fair as possible. We evaluate the performance of instance segmentation algorithms in many aspects, among which the most critical indicators are accuracy, running speed, and memory consumption.

The most common metric for accuracy is the mean Average Precision (mAP) used in Pascal VOC and its adjustment used in COCO. However, to understand mAP, we need to go through other metrics: Intersection-over-union (IoU), precision, and recall. We decide whether a prediction is correct or not with IoU or Jaccard Index. We define the IoU as the intersection of the predicted instance mask and the ground truth mask divided by their union. We consider a prediction as True Positive (TP) if the IoU > threshold and False Positive (FP) if IoU < threshold. The recall is the percentage of TPs found out from the ground truths. Precision is the percentage of correct predictions.

We can calculate AP per class now that we know the IoU, precision, and recall. Then, after calculating the AP per class, we can average them to get the mAP. We will use the simplified box example shown in figure 2.7a) to understand the AP calculation, where ground truths are in green and box predictions are in red and the IoU threshold is 0.5.

First, we sort the predictions based on the confidence score. If there is more than one prediction for a single object, the prediction having the highest IoU is considered TP,



(a) Example images

(b) PR curve

Figure 2.7: An example of two images 2.7a) containing ground truths in green and box predictions in red (Kumar, 2019), and 2.7b) shows their corresponding actual PR curve in blue and its approximation in red.

Detection	Confidence	TP/FP	Acc. TP	Acc. FP	Precision	Recall
P4	0.98	TP	1	0	1.0	0.33
P3	0.88	FP	1	1	0.5	0.33
P1	0.78	TP	2	1	0.66	0.66
P2	0.6	FP	2	2	0.5	0.66

Table 2.1: The table used to plot the PR curve 2.7b). It contains sorted detections according to their confidence, then the decision whether they are TP or FP and the precision and recall at each step.

and the rest are FPs. Next, we calculate the accumulated TP encountered from top to bottom, and we do the same for accumulated FP. Then, we calculate the precision and recall at each step as shown in table 2.1. Finally, we plot the precision-recall (PR) curve as in figure 2.7b) and the area under the curve is the AP. This metric captures the overall performance because it takes into account all other metrics like number of TP, FP, FN, precision, and recall.

In Pascal VOC, they calculated an approximation of this area. First, they interpolated the precision at each recall level by taking the maximum precision measured lower than that recall as shown in the red curve in figure 2.7b). Then, they took the mean of precision values at 11 equally spaced recall levels $[0, 0.1, \ldots, 1]$ (0 to 1 at a step size of 0.1). In our example this would be $(1+1+1+1+0.66+0.66+0.66+0+0+0+0)/11 = 0.5454 mAP$.

The mAP metric was modified in COCO to reward detectors with better localization. They calculated multiple mAPs with more than one IoU threshold and then averaged those mAPs to get the new mAP. For example, in Pascal VOC mAP, two predictions of IoU 0.6 and 0.9 would not change the metric. In COCO evaluation, the IoU thresholds

range from 0.5 to 0.95, with a step size of 0.05. They also report APs at fixed IoUs such as $IoU = 0.5$ and $IoU = 0.75$ written as AP^{50} and AP^{75}, respectively. Furthermore, they report mAP of specific mask sizes to differentiate accuracy across different levels.

Speed and memory consumption are also critical factors in instance segmentation. We measure speed with processing time and frames per second. The processing time is the needed time to process a standard resolution image. When we make speed comparisons of algorithms, we must ensure that we are using the same hardware as processing times could change massively between different hardware platforms.

Chapter 3

Instance segmentation with compact mask representations.

3.1 Motivation

When designing an instance segmentation algorithm, we want it to be as accurate, memory efficient, and fast as possible. Instance segmentation methods are complex, and there are many design choices; and one of the critical design choices is the mask representation. In our work, we will design a compact (or memory-efficient) mask to have an instance segmentation algorithm that can easily fit the vehicles' embedded hardware.

We discussed the four mask representations for instance segmentation in section 2.4: grid, point, polygon, and implicit representations. The grid representation is the most used and straightforward. However, it is not memory efficient, and we want to explore more efficient alternative representations. The point representation requires post-processing, and it is hard to get accurate masks with few points. The polygon and implicit representations are good candidates for a compact representation. The methods that used compact representations were PolarMask, ExtremeNet, and ESE-seg, which are polygon and implicit representation methods.

ExtremeNet (Zhou *et al.*, 2019b) used a keypoint detector to obtain the extreme points of an object. Extreme points are the outermost points of an object in each direction. With these points, they formed octagons, giving the object a rough mask, as shown in figure 3.1a. Although ExtremeNet is primarily an object detection method, the mask representation is the most compact representation with only eight parameters. We want to design a better mask with less or the same number of coefficients.

Polarmask (Xie *et al.*, 2020) used a polygon representation and performed a dense regression of the distances from the mask center to points on the outer contour in polar coordinates. This method performs well at a high number of points. However, the performance deteriorates at a low number of points, and the masks start to look polygon-like, as shown in figure 3.1b).

ESE-seg (Xu *et al.*, 2019) used an implicit representation and trained a network to predict a shape vector (a vector of coefficients), in which a numerical transform converts it into contour points in either the cartesian or polar representations as shown in figure

(a) ExtremeNet (b) Polarmask (c) ESE-seg

Figure 3.1: Example predictions of methods that use low number of parameters to represent the mask. Sub-figures 3.1a), 3.1b), and 3.1c) show an example prediction of ExtremeNet, Polarmask, and ESE-seg, respectively.

3.1c). The main advantage of this method is that it requires fewer parameters to represent the mask than the binary grid or polygon representations. However, ESE-Seg regresses the shape vector directly. We argue that the direct regression of the shape vector does not weigh each coefficient according to its impact on the mask and prevents the model from learning the actual data distribution.

Therefore, we propose an alternative training method in which the network outputs are passed through a differentiable shape decoder to obtain contour points. This differentiable shape decoder enables us to use the losses of polygon representation methods, e.g., PolarIOUloss (Xie *et al.*, 2019) and Chamfer loss (Fan *et al.*, 2017), and we train the network for its primary task. The gradients of these losses are back-propagated through the decoder, which automatically balances the weight of the different shape vector's coefficients. We base this chapter on our work in the paper "FourierNet: Compact Mask Representation for Instance Segmentation Using Differentiable Shape Decoders Riaz *et al.* (2020)," The rest of the chapter will explain our method.

3.2 Method

3.2.1 Architecture

We designed FourierNet to be an anchor-free, fully convolutional, single-stage network, as illustrated in figure 3.2. Anchor-free means that it does not classify anchor boxes to whether they have an object or not. As a result, the system is simple as we do not need to tune the anchor box hyperparameters, such as their size and positions, which highly influence the performance. Fully convolutional means that the network has only convolution and pooling operations and does not have the parameter-heavy fully connected layers at the end of the network. Moreover, single-stage means that it does not need an intermediate stage like object detection or semantic segmentation, as discussed in section 2.3. All these design choices make the network fast and straightforward.

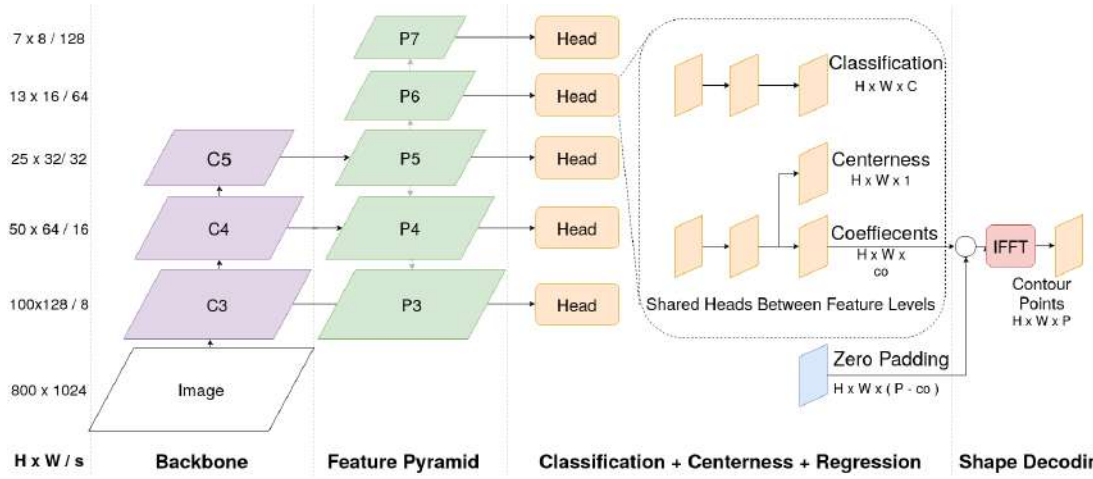


Figure 3.2: The FourierNet architecture (Note: The head shown in the figure is for polar representation only).

FourierNet has a CNN backbone, followed by a top-down feature pyramid network (FPN) (Lin *et al.*, 2017a) with lateral connections, in which we connect five heads with different spatial resolutions. These heads predict classification scores, centerness, and Fourier coefficients at each spatial location in the feature map. The classification branch predicts scores for each class. The centerness branch predicts the closeness of a feature point to the mask's center, and we explain it in section 3.2.4. Finally, the coefficients branch predicts the Fourier coefficients, which we will explain in the following mask representation section.

3.2.2 Mask representation

In FourierNet, we define the mask as a 2D Fourier series. If the value of that Fourier series at a specific position is higher than a threshold, it is considered foreground; otherwise, it is background. We can evaluate the Fourier series at different values depending on the representation to get the boundary points; however, this is costly. A faster way is to use the inverse fast Fourier transform (IFFT) to transform the signal from the frequency domain to the spatial domain. We can do this because the Fourier series coefficients are the fast Fourier transform (FFT) coefficients multiplied by the number of points. As the IFFT algorithm gives the same number of points as the coefficients, we pad the coefficients with zeros at high-frequency locations to get point-rich masks, this is shown in figure 3.2. The series can represent the boundary either in the cartesian or polar spaces, and the following sections will describe them in detail.

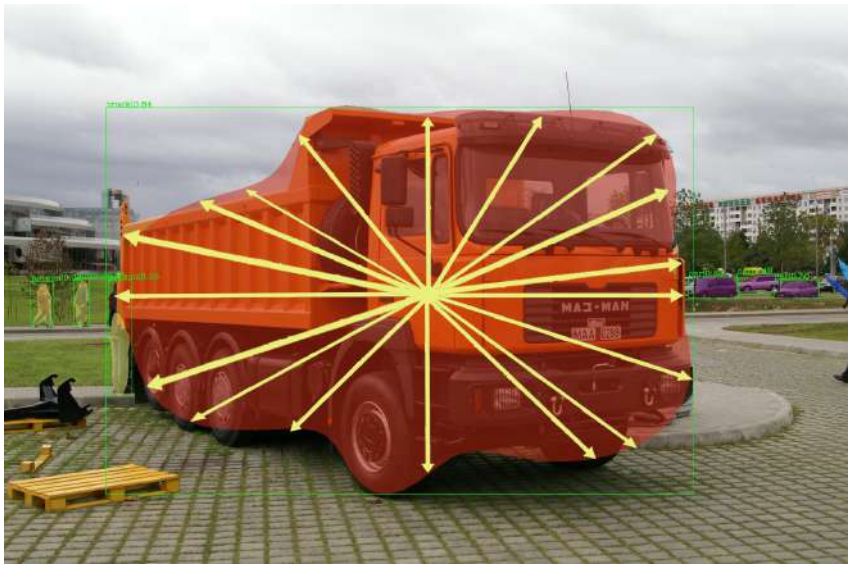


Figure 3.3: On the left object, 18 rays are extended by the lengths P_i from a feature point i (a potential center point). The contour points are the endpoints of these rays. Note that this figure is simplified for illustration purposes only (the mask has 90 contour points).

Polar representation

In the polar representation, for each feature point i near the contour's center, we define the mask boundary as

$$P_i(\theta) = \sum_{k=-N}^{N} \left(c_{k,i} \cdot e^{jk\theta} \right), \tag{3.1}$$

where $P_i \in \mathbb{R}$ is the distance from the feature point i to the mask boundary at the angle θ from the x-axis, $c_{k,i} \in \mathbb{C}$ is the k-th exponential form Fourier series coefficient, N is the number of frequencies used in the Fourier series. The Fourier series coefficients determine the shape of the function, and the network predicts these coefficients. For real-valued functions, the Fourier series has a property where $c_{k,i} = c^*_{-k,i}$ (* denotes complex conjugation.), this property helps in reducing the number of parameters the network needs to predict. To determine P_i, we can either evaluate 3.1 at different θ values or apply an *Inverse Fast Fourier Transform* (IFFT) to the coefficients predicted by the network (figure 3.2). The inverse discrete Fourier transform is an invertable, linear transformation defined by

$$p_{m,i} = \frac{1}{M} \sum_{k=0}^{M-1} x_{k,i} \, e^{\frac{j2\pi km}{M}}, \tag{3.2}$$

where $p_{m,i}$ is the m_{th} point in P_i, M is the number of desired points, and $x_{k,i}$ is the k_{th} coefficient of X_i, which are the Fourier coefficients padded with zeros at high frequencies. This is because in most cases, we want to predict more points than Fourier coefficients.

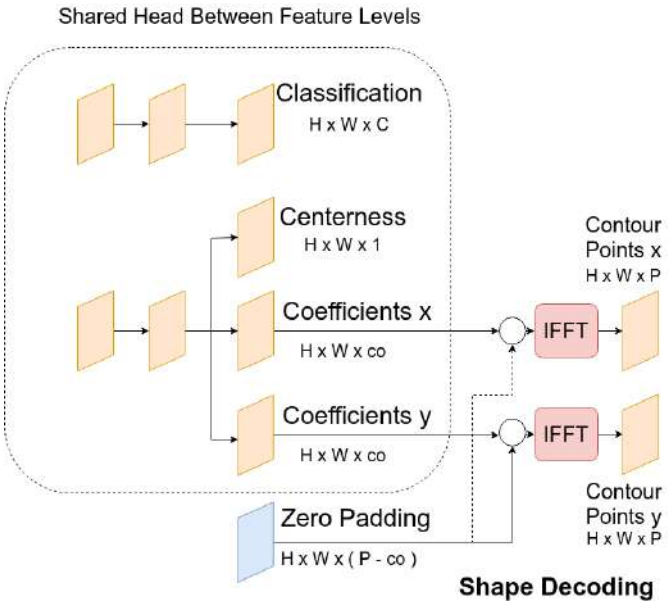


Figure 3.4: The FourierNet head for cartesian representation.

This is done to equalize the dimensions before and after the IFFT. Note that the IFFT algorithm is differentiable and therefore the training is done directly on P_i and thus the name *differentiable shape decoder*. The resultant $P_i = \{p_{0,i}, p_{1,i}, ..., p_{M-1,i}\}$ can be seen as the length of equidistant rays extending from a center point i, as shown in figure 3.3. The angle between the rays $\Delta\theta$ is constant and defined by $360°/M$.

The emerged contour will be an approximation to the ground truth contour even with a high number of rays, and we will discuss the effectiveness of this approximation in section 3.2.3. In addition, because of this approximation, we need to take care of unique cases during training. One case is when the ray intersects more than one boundary point; here, we select the point with the longest distance to the center. Furthermore, when the feature point i is outside or on the contour's boundary, we assign a constant $\varepsilon = 10^{-6}$ to the ray that does not have intersection points.

Cartesian representation

The polar representation generates star-shaped masks since, for each angle, there is only one possible ray length. To represent arbitrary masks, we can use cartesian coordinates. We represent the mask boundary as two functions for cartesian representation, one for x and the other for y. We define the functions as follows:

$$P_i^x(t) = \sum_{k=-N}^{N} \left(c_{k,i}^x \cdot e^{jkt} \right), \tag{3.3}$$

$$P_i^y(t) = \sum_{k=-N}^{N} \left(c_{k,i}^y \cdot e^{jkt} \right), \tag{3.4}$$

where $P_i^x(t)$ and $P_i^y(t)$ are distances from the feature point i in the x and y directions, respectively. $c_{k,i}^x$ and $c_{k,i}^y$ are the k-th exponential form Fourier series coefficients which we make the FourierNet head to predict. t is a variable that tells which part of the contour the point is and ranges from 0 to 2π. To get a point the equations 3.3 and 3.4 are evaluated at the same t value. Figure 3.4 shows the structure of the FourierNet head for cartesian representation. An IFFT is applied to each of these branches separately. We pad the output tensor with zeros for cases where contour points are more than Fourier coefficients, just like in the polar representation.

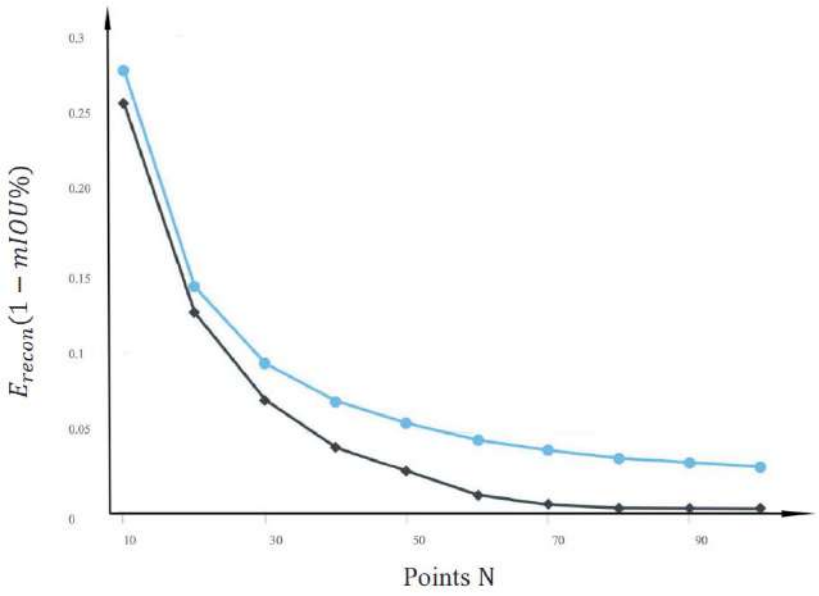


Figure 3.5: The reconstruction loss against the number of parameters for the polar (in blue), and the cartesian (in black) representations (Xu *et al.*, 2019). We can see that the error goes almost to zero for the cartesian representation at high number of points, while the polar's error stays constant at about 4%.

3.2.3 Mask representation upper bound

The mask representation we use in our work can be only an approximation of the ground truth. The cartesian representation is limited to single mask instances. On the other hand, the polar representation is limited to star-shaped masks. So we should measure how good these approximations are. One way to do this is to reconstruct the mask with the representation and compare it to the ground truth masks of a dataset. Xu *et al.* (2019)

did this experiment on the PASCAL VOC 2012 dataset, where they reconstructed the masks using both representations when sampling them with a different number of points. Figure 3.5 reports the reconstruction error as a function of the number of points. The cartesian representation has a lower reconstruction error compared to the polar representation when having the same number of points. In addition, The cartesian error almost goes to zero after using 90 points, while the polar error stops at around 5%. However, the cartesian optimization problem is a 2D, while the polar optimization problem is a 1D which should be easier to solve. In the experiment section, we will compare the two.

3.2.4 Centerness

Centerness is a term that measures the closeness of a feature point to the center of a mask, and we use it to filter out weak detections. This was an observation by Tian *et al.* (2019), where they found that detections at instance edges have bad quality. During inference, we multiply this value with the classification score to keep the locations, which could produce the best detection. We utilize *polar centerness* and *normalized centerness* in the case of polar representation and *gaussian centerness* in the case of cartesian representation. Both are detailed in the following sections, respectively.

Polar centerness

Polar Centerness (PC) (Xie *et al.*, 2019) is defined for the i_{th} feature point as

$$PC_i = \sqrt{\frac{\min(p_{0,i}, p_{1,i}, ..., p_{N-1,i})}{\max(p_{0,i}, p_{1,i}, ..., p_{N-1,i})}}, \tag{3.5}$$

where $p_{n,i}$ are the ray lengths. This metric will have low values if the feature point is near the boundary, and this would filter out bad detections as was shown by Tian *et al.* (2019). On the other hand, this metric will be low if the object's mask shape is not circular, and since we multiply it by the classification score, it will lower the probability of predicting such objects. As a result, PolarMask introduced a hyperparameter called *Centerness Factor* (CF) to overcome this problem, which is a constant added to all centerness values. We think that this offset defeats the purpose of centerness, since it artificially raises confidence and sometimes even exceeds 1. Moreover, it does not explicitly solve the problem of low centerness of non-circular objects. Therefore, we introduce *Normalized Centerness* (*NC*) which is defined for a feature point *i* by

$$NC_i = \frac{PC_i}{PC_{\max}}, \tag{3.6}$$

where PC_{max} is the polar centerness of the *center of mass* of an instance. The maximum value of the NC_i is clamped to one, when the center of mass does not have the highest

polar centerness value.

Gaussian Centerness

In the case of centerness for cartesian representations, we can not adopt equation 3.5 directly. Accordingly, we apply a Gaussian distribution to represent the probability of a point being at the object's center. For the ith feature point having the location (m,n) in the feature map, Gaussian centerness (GC) is defined as

$$GC = e^{-\alpha(\frac{m-\mu_x}{\sigma_x})^2} e^{-\alpha(\frac{n-\mu_y}{\sigma_y})^2}, \tag{3.7}$$

where μ_x and μ_y are the means (center points), and σ_x and σ_y are the standard deviations of a mask instance in x and y directions, respectively, and α is a hyperparameter used for controlling the decay rate. Note that we multiply the two Gaussians, which enforces a probability of 1 only if both m and n are at the object's center. On all the other locations, the decaying functions' product reduces the centerness depending upon the standard deviation in both x and y directions of the mask instance. Notice that GC solves the problem of low centerness for non-circular objects, and therefore the centerness factor can be completely avoided.

3.2.5 Loss functions

The overall loss function comprises of four components, which is defined as:

$$L_{total} = L_{cls} + L_{cent} + L_{mask} + L_{box}. \tag{3.8}$$

We use the *focal loss* (Lin *et al.*, 2017b) for the classification loss L_{cls}. The focal loss is given by

$$FL(p) = \begin{cases} -\alpha(1-p)^{\gamma}\log(p) & \text{if correct class} \\ -(1-\alpha)(p)^{\gamma}\log(1-p) & \text{otherwise} \end{cases} \tag{3.9}$$

where p is the class probability predicted by the network, γ is the focusing parameter to make the network focus on hard or easy examples, and α is the balancing factor which helps in class imbalance. For centerness loss L_{cent} in both *polar centerness* and *gaussian centerness*, we employ the *binary cross entropy* which is given by

$$CE(p) = -(q \cdot \log(p) + (1-q) \cdot \log(1-p)) \tag{3.10}$$

where q is the target centerness value and p is the predicted centerness value. For mask loss L_{mask}, we utilize two different loss functions for polar and cartesian representations. For cartesian representation, we employed both an index-wise *smooth L1* loss and *cham-*

fer distance loss (Fan *et al.*, 2017). The chamfer distance loss is defined as

$$CD = \sum_{a \in S_1} \min_{b \in S_2} \| D(a,b) \|_2^2 + \sum_{b \in S_2} \min_{a \in S_1} \| D(a,b) \|_2^2, \tag{3.11}$$

where S_1 and S_2 are the sets of predicted contour points and ground truth contour points respectively, a and b are elements (individual contour points (x,y)) of the sets S_1 and S_2 respectively and $D(a,b)$ is the euclidean distance between any two points a and b, respectively. We normalize the chamfer distance by dividing it by the average of the height and width of the ground truth bounding box. Without the normalization, the chamfer distance becomes exceptionally large, which leads to overflows and exploding gradients. Moreover, normalization avoids the problem of manually weighing classification, centerness and mask losses. In case of the polar representation, we adopt the *Polar IOU loss* from (Xie *et al.*, 2019), which is defined as

$$PolarIOU = \log\Big(\frac{\sum_{i=0}^{m} min(d_i, d_i^*)}{\sum_{i=0}^{m} max(d_i, d_i^*)}\Big) \tag{3.12}$$

where d_i and d_i^* are the i-th predicted and the ground truth rays constructing the mask. Here, the rays are compared index-wise. The *Polar IOU loss* is able to automatically keep the balance between classification loss and regression losses. We use the *IOU loss* (Yu *et al.*, 2016) for the bounding box loss L_{box}. Note that the bounding box branch is an optional branch and therefore not explicitly shown in the figure 3.2.

3.3 Experiments

We conducted the experiments on the COCO 2017 benchmark (Lin *et al.*, 2014). We based our work on the mmdetection framework. Unless otherwise stated, we did all the experiments using a pre-trained ResNet-50 (He *et al.*, 2015) on ImageNet (Krizhevsky *et al.*, 2012). We trained the networks for 12 epochs with an initial learning rate of 0.01 and a mini-batch of 4 images. The learning rate was reduced by a factor of 10 at epochs 8 and 11. We used Stochastic gradient descent (SGD) with momentum (0.9) and weight decay (0.0001) for optimization. We resized the input images to 1280×768 pixels.

3.3.1 Cartesian representation vs. polar representation

We compared various networks trained on the cartesian representation using smooth L1 loss and chamfer distance loss and the polar representation with the PolarIOU loss. The networks trained with Chamfer distance loss were first pre-trained for one epoch on smooth L1 loss as a warm-up. This warm-up provides a good initialization, since chamfer loss greatly benefits from elliptical predictions at the start. In all the experiments, we selected $\alpha = 10$, which provides a reasonable balance between a high probability for a

point at the center of the object and low values at the mask edges.

Coeff.	Loss mask	Loss centerness	mAP
8	Smooth L1	Gaussian	13.6
36	Smooth L1	Gaussian	13.5
8	Chamfer	Gaussian	22.9
36	Chamfer	Gaussian	22.4
8	Polar IOU	Polar	26.9
36	Polar IOU	Polar	28.0

Table 3.1: Comparison of mAP for the cartisian and polar representations

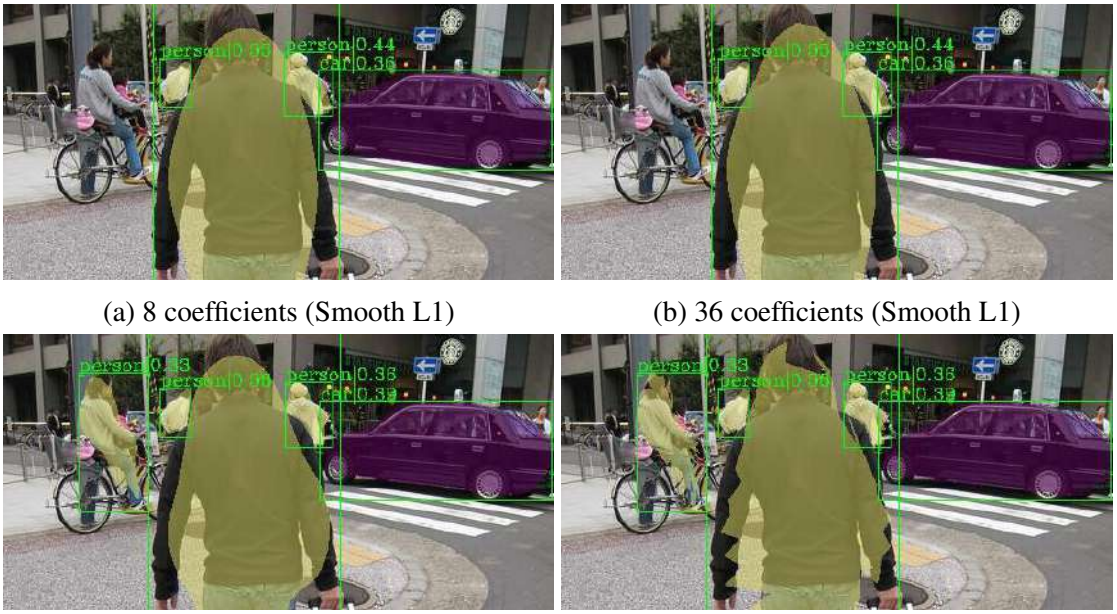
Figure 3.6 illustrates the masks generated by the network using cartesian representation, and table 3.1 reports the quantitative results. The network with smooth L1 loss performs worst as it only makes ellipse-like masks. Moreover, the difference between 8 and 36 coefficients is insignificant, as shown in figures 3.6a) and 3.6b). Counter-intuitively, the cartesian masks trained with Chamfer loss and 8 coefficients are smoother and have a better IOU than the masks with 36 coefficients. Figure 3.6d) shows that this is because of undesired oscillations when using 36 coefficients, which makes the contour worse. One possible reason could be that the gradients are very low for the higher frequency coefficients during training, because they affect the output very little. This eventually leads to under-trained higher frequency coefficients, which show fluctuating masks. However, this hypothesis needs further investigation, which we did not investigate in our work. The best performance in the cartesian space with 22.9 mAP falls short of the polar representation, which achieves 28.0 mAP. Since the polar representation showed superior performance, we will employ the polar representation in the rest of the work.

3.3.2 Ablation study

All the experiments done in this ablation study section adopt the polar representation for masks.

Coefficients regression (CR) vs. Differentiable shape decoding (DSD)

The main goal of this section is to show that Differentiable shape decoding is vital to training explicit representation methods, and unweighted direct coefficient regression focuses during training equally on all coefficients, which is not optimal for shape decoders. On the contrary, when trained on contour points, the optimizer can inherently learn to prioritize the more essential frequency coefficients of the Fourier series and achieve automatic weight balancing. We trained a network with 18 coefficients to verify this hypothesis and regressed the coefficients directly using a smooth L1 loss. The network trained with the smooth L1 loss attained an mAP of 5.3 (table 3.2), which is inferior



(a) 8 coefficients (Smooth L1)

(b) 36 coefficients (Smooth L1)

(c) 8 coefficients (Chamfer distance)

(d) 36 coefficients (Chamfer distance)

Figure 3.6: Examples of the network using cartesian coordinates. Sub-figure 3.6a and 3.6b are trained using smooth L1 loss. Sub-figure 3.6c and 3.6d are trained using chamfer distance loss.

compared to the 26 mAP of a similar network trained on contour points (table 3.2 and figure 3.10) and this experiment validates our initial intuition. Moreover, the qualitative results of CR showed out-of-size masks because of mistakes in low-frequency coefficients, where the lower frequency coefficients of a Fourier series have a higher impact on the contour. ESE-Seg (Xu *et al.*, 2019) used CR and compared various function

Method	mAP	AP_{50}	AP_{75}
Coefficient Regression	5.3	14.9	3.1
Differentiable Shape Decoding	26	46.6	25.8

Table 3.2: Coefficients regression (CR) vs. Differentiable shape decoding (DSD)

approximators, including the Fourier series. They reported that the best function approximator is the Chebyshev polynomial and argued that this is because they have the best numerical distribution. However, we showed that we could better perform using the Fourier series and the DSD because it does automatic weight balancing and overcomes the unsuitable numerical distribution problem. In addition, we argue that the Chebyshev polynomials have discontinuities that would affect the mask's shape. Figure 3.8 shows these discontinuities that look unrealistic compared to the masks of our method.

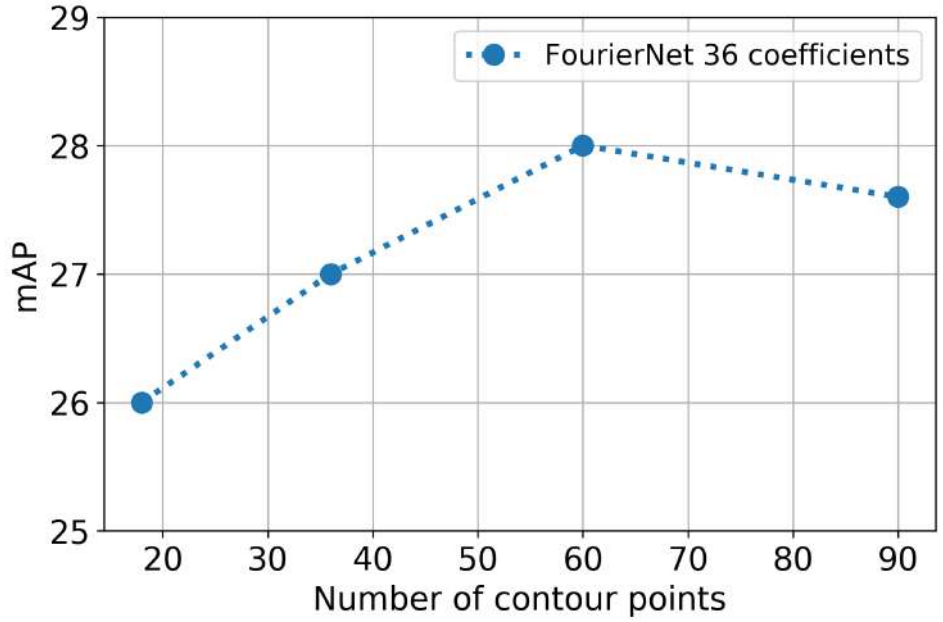


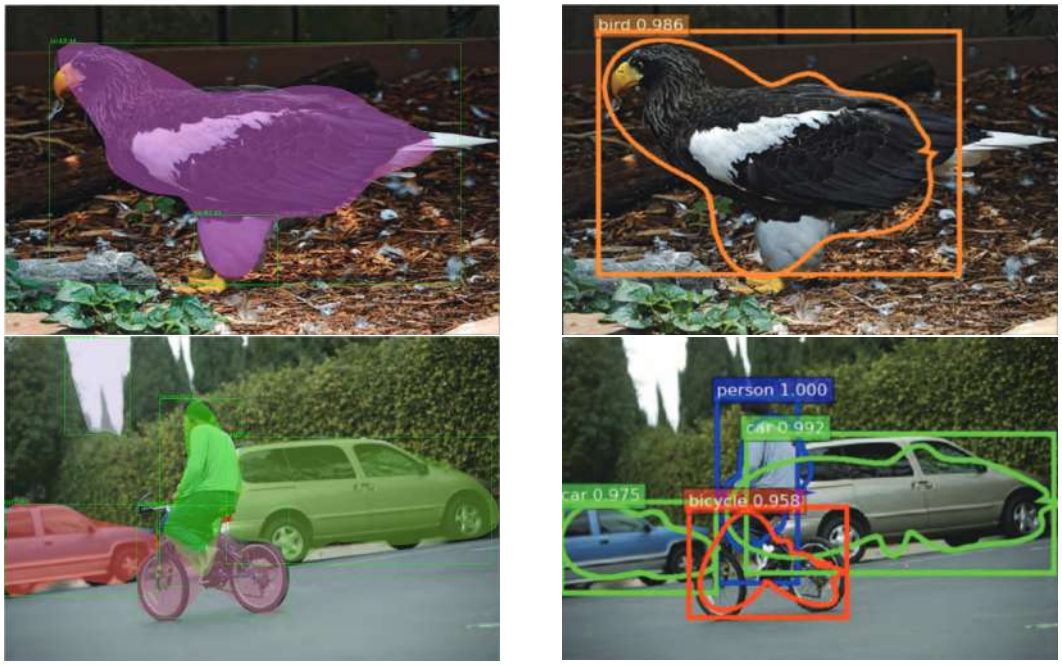
Figure 3.7: The relationship between the mAP and the number of sampled points. The FourierNet in this experiment has 36 coefficients of Fourier series and ResNet-50 backbone.

Number of contour points

We trained multiple FourierNets having 18, 36, 60, and 90 contour points and 36 complex coefficients (72 parameters). Figure 3.7 shows that more contour points generally lead to a higher mAP until it saturates, and then the performance deteriorates. The FourierNet with 90 points has a lower mAP (27.6) than a FourierNet with 60 points (28.0), and a possible reason could be that the added complexity (in terms of contour points) makes the problem harder for the optimizer to learn. Furthermore, while more contour points seem more appealing for large and complicated masks, for smaller objects. it means adding unwanted complexity, which could lower AP_s if not learned correctly, leading to an overall negative effect on performance.

Number of coefficients

From the results of the previous section, we choose a FourierNet with 60 contour points for this study. Figure 3.10 illustrates the relationship between the networks' accuracy and the number of parameters representing the mask. We generated the FourierNet curve using one network, but we tested it multiple times. Each time we take a number of the predicted coefficients with the lowest frequencies and set the high frequencies to zeros. We made this step because we trained networks with a low number of coefficients, and they gave similar results to networks with a high number of coefficients and suppressed



(a) FourierNet-640 (b) ESE-Seg-416

Figure 3.8: A qualitative comparison between FourierNet-640 and ESE-Seg-416.

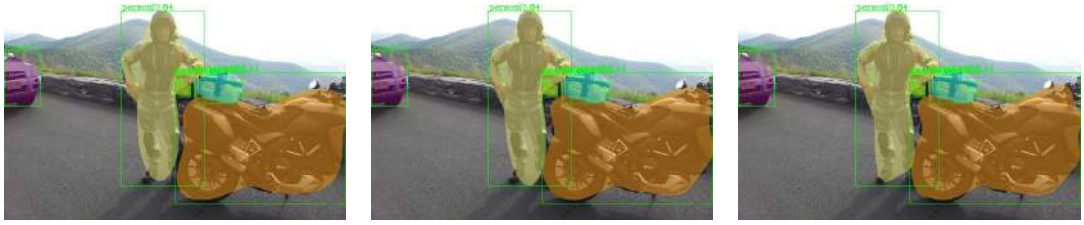
high frequencies. This finding helps to find a suitable number of coefficients for a specific application, and in our case, it reduced the number of experiments.

As the number of parameters increases, the mAP sharply increases until around 18 parameters, and after 36 parameters, it saturates. Our method gives the best performance compared to all compact mask representations, especially when using a low number of coefficients. We also observed that the FourierNets with 18, 36, and 90 points showed the same curve trend as FourierNet with 60 points, so we did not plot them. Furthermore, We visualize in figure 3.9 the network outputs with a different number of suppressed higher frequencies. We obtain smooth contours if we use a low number of coefficients, and when we utilize only two coefficients, all the predictions become ellipses. Figure 3.9 illustrates how suppressing higher frequencies affect the complexity (variations in ray lengths to angles) of the mask of a particular object. A sinusoidal curve (ellipse) is observed for two coefficients, while for 20 coefficients, we see a cumulative effect of multiple sinusoidal components.

Polar centerness (PC) vs. Normalized centerness (NC)

We introduced NC to tackle the effect of low PC values, as explained in section 3.2.4. We trained two networks with 90 contour points and 36 coefficients on NC and PC. The results in table 3.3 show that NC is better than PC when we set the CF to zero, which

(a) 2 coeff. (4 parameters.) (b) 3 coeff. (6 parameters.) (c) 5 coeff. (10 parameters.)



(d) 7 coeff. (14 parameters.) (e) 10 coeff. (20 parameters.) (f) 36 coeff. (72 parameters.)

Figure 3.9: Comparison between the predicted mask of using varying number of Fourier coefficients using polar representation.

means it is generally a better centerness metric. However, we still need to use the CF hyperparameter to obtain the best performance.

Method	CF	mAP	AP_{50}	AP_{75}
Polar	0	26.3	42.8	**27.7**
Normalized	0	**27.0**	**47.8**	26.9
Polar	0.5	**27.7**	46.4	**28.6**
Normalized	0.5	27.0	**47.9**	26.9

Table 3.3: Polar centerness vs. Normalized polar centerness

3.3.3 Comparison to state-of-the-art

We trained a FourierNet-640 with an image resolution of 640 × 360 to compare with an ESE-Seg-416 (Xu *et al.*, 2019), which was trained with a resolution of 416 × 416. With a comparable backbone and the same number of parameters, our result is 2.7 mAP higher, and it runs in real-time as we see in table 3.4 and figure 3.9. We trained a FourierNet with a ResNeXt101 backbone (Xie *et al.*, 2017), 90 contour points, and 36 coefficients to compare against other state-of-the-art methods and table 3.4 shows the quantitative results. Compared to ExtremeNet (Zhou *et al.*, 2019b), which uses an eight parameter mask, our performance is better, especially with AP_L and AP_{75}, with an increase of 6.1

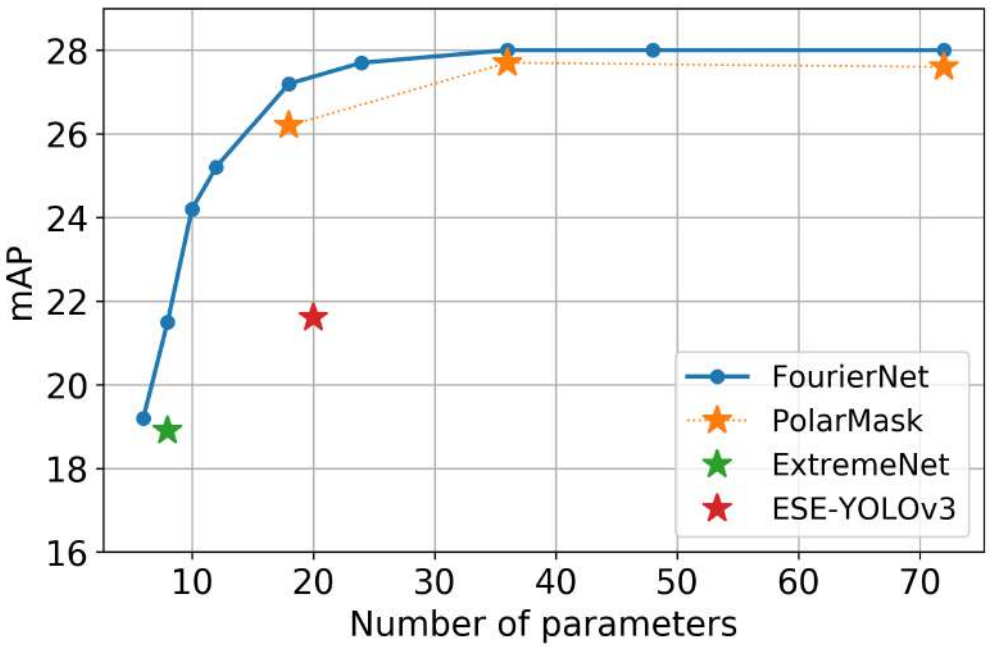


Figure 3.10: The Fouriernet in this experiment has a **Resnet-50** backbone and **60 contour points**. The number of parameters are the complex (real+imaginary) coefficients of the fourier series i.e 36 coefficients = 72 parameters. The PolarMask network used in this experiment has the same backbone as well.

and 7.4, respectively. These results mean that our mask quality is superior when using a few parameters. FourierNet is comparable to PolarMask when using the same number of parameters, with a slight loss in speed due to the IFFT. However, the qualitative results are visually better with smoother contours. Our method is generally comparable to polygon methods but falls short in performance compared to binary grid methods.

3.4 Conclusion

FourierNet is a single-stage anchor-free method for instance segmentation. It uses a novel training technique with IFFT as a differentiable shape decoder that decodes the mask into a cartesian or a polar representation. Theoretically, we showed that the Cartesian representation has a higher upper bound than the polar representation; however, it was hard to train and gave a lower performance in experiments. We believe that the reason for this is that the polar representation is a 1D problem, which is easier to solve than the 2D cartesian optimization problem.

Moreover, we could obtain a compact mask representation with low frequencies because they contain most of the mask's information. Therefore, FourierNet outperformed all methods which use less than 20 parameters quantitatively and qualitatively. Fur-

thermore, compared to object detectors, FourierNet can yield better approximations of objects than boxes using slightly more parameters. Furthermore, we showed that our normalized centerness is generally a better centerness metric than the polar centerness when we do not use the centerness factor. Finally, our FourierNet-640 achieves a real-time speed of 26.6 FPS, and the FourierNet with ResNext-101 achieved comparable results to other polygon and implicit representations. We hope this method can inspire the use of differentiable decoders in other applications.

Method	B.Bone	Rep.	Par.	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L	FPS	GPU
two stage											
Mask RCNN (He *et al.*, 2017)	RX-101	grid	784	37.1	60.0	39.4	16.9	39.9	53.5	5.6	1080Ti
PANet (Liu *et al.*, 2018)	RX-101	grid	784	**42.0**	**65.1**	**45.7**	22.4	**44.7**	**58.1**	-	-
HTC (Chen *et al.*, 2019)	RX-101	grid	784	41.2	63.9	44.7	**22.8**	43.9	54.6	2.1	TitanXp
one stage											
ESE-Seg-416 (Xu *et al.*, 2019)	DN-53	implicit	20	21.6	**48.7**	22.4	-	-	-	**38.5**	1080Ti
FourierNet-640	R-50	implicit	20	**24.3**	42.9	**24.4**	6.2	25.9	42.0	26.6	2080Ti
ExtremeNet (Zhou *et al.*, 2019b)	HG-104	polygon	**8**	18.9	44.5	13.7	**10.4**	20.4	28.3	3.1	-
FourierNet	RX-101	implicit	**8**	**23.3**	**46.7**	**21.1**	10.3	**25.2**	**34.4**	**6.9**	2080Ti
EmbedMask (Ying *et al.*, 2019)	R-101	grid	†	**37.7**	**59.1**	**40.3**	**17.9**	**40.4**	**53.0**	13.7	V100
YOLACT-700 (Bolya *et al.*, 2019)	R-101	grid	†	31.2	50.6	32.8	12.1	33.3	47.1	**23.4**	TitanXp
PolarMask (Xie *et al.*, 2019)	RX-101	polygon	36	32.9	55.4	33.8	15.5	35.1	46.3	7.1*	2080Ti
FourierNet	RX-101	implicit	36	30.6	50.8	31.8	12.7	33.7	45.2	6.9	2080Ti

Table 3.4: Comparision with state-of-the-art for instance segmentation on COCO test-dev. † The number of parameters are dependent on the size of the bounding box cropping the pixel embedding or mask prototype. * speed tested on our machines.

Chapter 4

Seeing implicit neural representations as Fourier series

4.1 Introduction

In FourierNet, the mask representation we used was limited to instances with single star-shaped masks. To solve this problem, we changed our mask design so it is a parametrized super level set by a 2D Fourier series. However, we found an interesting connection between the Fourier series and implicit neural representations during our work, which extended our work even further in mask representations for instance segmentation, as we will see in detail in chapter 5. This chapter will talk about the connection between the Fourier series and INR.

Implicit neural representation is a field of research that studies replacing traditional discrete signal representations with neural networks that map the input domain of the signal to the information at the input location. For example, instead of representing images as discrete grids of pixels, we make a coordinate-based MLP, as illustrated in figure 4.1a), to map the coordinates of a specific pixel to its color. Likewise, instead of representing 3D shapes as voxel grids or meshes, we map point coordinates to their occupancy or physical property, as shown in figure 4.1b). INRs are approximations of those signals, and the goal is to get the approximations as accurate as possible.

An advantage of INRs is that they are not coupled to the spatial resolution (e.g., voxel size in a 3D scene), and theoretically, they have infinite resolution. Therefore, these representations are naturally suited to high-dimensional signals and heavy memory consumption applications. Furthermore, they are differentiable, and as a result, they are suitable for gradient-based optimization and machine learning. In addition, the usage of INRs for images (Henzler *et al.*, 2020; Stanley, 2007), volume density (Mildenhall *et al.*, 2020), and occupancy (Mescheder *et al.*, 2019) enhanced the performance on various tasks such as shape representation (Chen and Zhang, 2019; Deng *et al.*, 2020; Genova *et al.*, 2019, 2020; Jiang *et al.*, 2020; Michalkiewicz *et al.*, 2019; Park *et al.*, 2019), texture synthesis (Henzler *et al.*, 2020; Oechsle *et al.*, 2019a), and shape inference from images (Liu *et al.*, 2020, 2019). With all of these advantages, we wanted to use them to represent masks for instance segmentation.

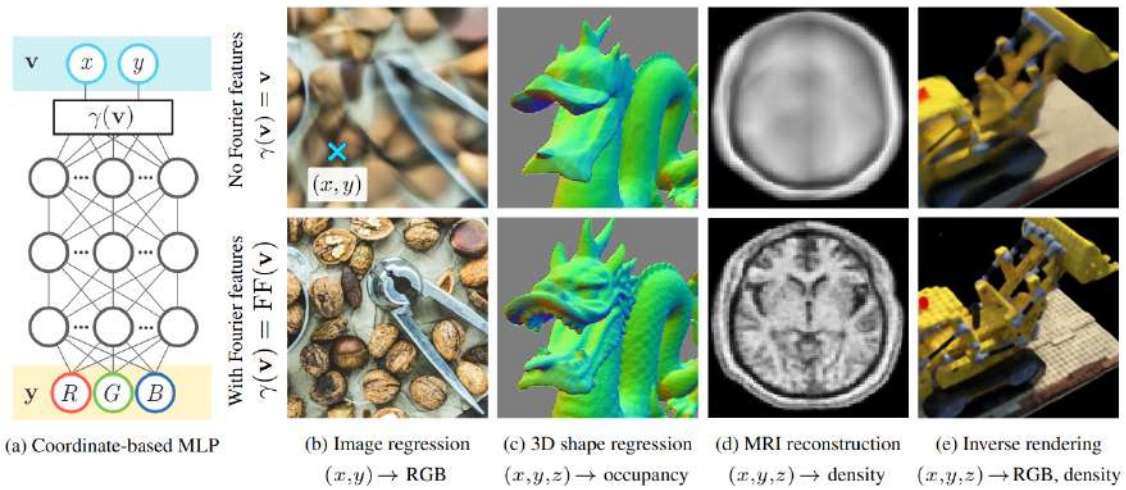


Figure 4.1: a) A visualization of an MLP used to represent images, where it maps the pixels' coordinate to thier color. b-e) show different tasks that use INRs to represent the signal. The top row shows results where the coordinates are passed directly to the MLP, and the bottom row shows results with a Fourier feature mapping. Fourier features enable the MLP to represent higher frequency details (Tancik *et al.*, 2020).

Early INR architectures lacked accuracy in high-frequency details. Sitzmann *et al.* (2020) proposed Sinusoidal Representation networks (SIRENs) to solve this problem. SIRENs use sinusoidal instead of ReLU activations in MLPs, while taking into account weight initialization to enable stable training. They argued that sinusoidal activations work better than ReLU activations, because ReLU networks are piecewise linear, and their second derivative is zero. As a result, they cannot model the data contained in higher-order derivatives of signals.

At the same time, Mildenhall *et al.* (2020) proposed using positional encoding to enable the networks to learn high-frequency information, as shown in the second row of figure 4.1. The positional encoding uses a heuristic sinusoidal mapping to input coordinates before passing them through a ReLU network. In a consequent work, Tancik *et al.* (2020) explored the general Fourier mapping and explained why it worked using a Neural Tangent Kernel (NTK) framework (Jacot *et al.*, 2018), they found out that the Fourier mapping transforms the NTK into a shift-invariant kernel. Furthermore, modifying the mapping parameters enables tuning the NTK's spectrum, controlling the range of frequencies the network can learn.

They also showed that a random Fourier mapping with low standard deviation learns only low frequencies of the signal. On the other hand, a high standard deviation lets the network learn high frequencies only, which leads to over-fitting. Therefore, they recommended a linear search to find the optimal value of the standard deviation for the corresponding task. They also showed that increasing the number of parameters in the

mapping improves the performance constantly. However, we wonder if random Fourier mapping is the optimal mapping? Will the performance be saturated if we continue to increase the mapping parameters? What is the difference between SIRENs and Fourier mapping? Moreover, is there a way to avoid over-fitting when training networks using Fourier mapping? Finally, can we use INR to represent masks in instance segmentation?

In this chapter, we will answer these questions. First, we determined the d-dimensional Fourier series's trigonometric form and showed that it is precisely a single perceptron with an integer lattice mapping applied to its inputs. The weights of that perceptron are the Fourier series coefficients. As the Fourier series can theoretically represent any periodic signal, this perceptron can represent any periodic signal if it has an infinite number of frequencies in its mapping. Then, we explored the mathematical connection between Fourier mappings and SIRENs and showed that a Fourier mapped perceptron is structurally like a hidden layer SIREN. However, in the SIREN case, the mapping is trainable, and it is represented in the amplitude-phase form instead of the sine-cosine form in the case of Fourier mappings.

Moreover, we modified the progressive training strategy of Lin *et al.* (2021), where we train the lower frequencies in the initial training phase and gradually add the higher frequency components as the training progresses. As a result, we show that our progressive training strategy avoids the problem of over-fitting. Finally, the results of this work gave good signs that it is possible to use INR for mask representations for instance segmentation. The work in this chapter is based on our paper "Seeing implicit neural representations as Fourier series" Benbarka *et al.* (2022).

4.2 Related work

Inspired by INRs' recent success, by outperforming grid-, point- and mesh-based representations (for the first time in 2018 (Park *et al.*, 2019),(Mescheder *et al.*, 2019),(Chen, 2019)), many works based on INRs achieved state-of-the-art results in 3D computer vision (Atzmon and Lipman, 2020; Gropp *et al.*, 2020; Jiang *et al.*, 2020; Peng *et al.*, 2020; Chabra *et al.*, 2020; Sitzmann *et al.*, 2020). Moreover, impressive results are obtained across different input domains, e.g., from 2D supervision (Sitzmann *et al.*, 2019; Niemeyer *et al.*, 2020; Mildenhall *et al.*, 2020), 3D supervision (Saito *et al.*, 2019; Oechsle *et al.*, 2019b), to dynamic scenes (Niemeyer *et al.*, 2019) which can be represented by space-time INR.

In early architectures, there was a lack of accuracy in fine details of signals. Mildenhall *et al.* (2020) proposed positional encodings to tackle this problem, then Tancik *et al.* (2020) further explored positional encodings in an NTK framework, showing that mapping input coordinates to a representation close to the actual Fourier representation before passing them to the MLP lead to a good representation of the high-frequency details. Furthermore, they showed that random Fourier mappings achieved superior results than if one takes the simple positional encoding. Sitzmann *et al.* (2020) also attempted to

solve the problem of getting high-frequency details. They proposed SIRENs and demonstrated that SIRENs are suited for representing complex signals and their derivatives. In both solutions, they used a variant of Fourier neural networks (FNN) for the first layer of the MLP. FNN are neural networks that use either sine or cosine activations to get their features (Liu, 2013). The first attempt to build an FNN was by Gallant and White (1988). They proposed a one-layer hidden neural network with a cosine squasher activation function and showed if they hand-wire certain weights, it will represent a Fourier series. Silvescu (1999) proposed a network that did not resemble a standard feedforward neural network. However, they used a cosine activation function to get the features. Liu (2013) introduced the general form for Fourier neural networks in a feedforward manner. They also proposed a strategy to initialize the frequencies of the embedding, which helped for convergence. Our work will show another way to initialize the embedding, which results in a neural network that is precisely a Fourier series.

4.3 Method

4.3.1 Integer lattice mapping

This section explains how a perceptron with an integer lattice Fourier mapping applied to its inputs is equivalent to a Fourier series. First, we present the Fourier mapped perceptron equation and then link it to the Fourier series's general equation. The fundamental building block of any neural network is the perceptron, and it is defined as

$$\mathbf{y}(\mathbf{x}, \mathbf{W}', \mathbf{b}) = g(\mathbf{W}' \cdot \mathbf{x} + \mathbf{b}). \tag{4.1}$$

Here $\mathbf{y} \in \mathbb{R}^{d_{out}}$ is the perceptron's output, $g(\cdot)$ is the activation function (usually nonlinear), $\mathbf{x} \in \mathbb{R}^{d_{in}}$ is the input, $\mathbf{W}' \in \mathbb{R}^{d_{out} \times d_{in}}$ is the weight matrix, and $\mathbf{b} \in \mathbb{R}^{d_{out}}$ is the bias vector. Now, if we let $g(\cdot)$ to be the identity function and apply a Fourier mapping to the input we get

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W} \cdot \gamma(\mathbf{x}) + \mathbf{b}, \tag{4.2}$$

where $\gamma(x)$ is the Fourier mapping defined as

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi \mathbf{B} \cdot \mathbf{x}) \\ \sin(2\pi \mathbf{B} \cdot \mathbf{x}) \end{pmatrix}. \tag{4.3}$$

$\mathbf{W} \in \mathbb{R}^{d_{out} \times 2m}$ is the weight matrix after applying the mapping, $\mathbf{B} \in \mathbb{R}^{m \times d_{in}}$ is the Fourier mapping matrix, and m is the number of frequencies. Equation 4.2 is the general equation of a Fourier mapped perceptron, and we will relate it to the Fourier series's general equation.

A Fourier series is a weighted sum of sines and cosines with incrementally increasing

frequencies that can reconstruct any periodic function when its number of terms goes to infinity. In applications that use coordinate-based MLPs, the functions we want to learn are not periodic. However, their inputs are naturally bounded (e.g., height and width of an image). Accordingly, it doesn't harm if we assume that the input is periodic over its input's bounds to represent it as a Fourier series. We will explain later why this assumption has many advantages. A function $f : \mathbb{R}^{d_{in}} \to \mathbb{R}^{d_{out}}$ is periodic with a period $p \in \mathbb{R}^{d_{in}}$ if

$$f(\mathbf{x} + \mathbf{n} \circ \mathbf{p}) = f(\mathbf{x}) \quad \forall \mathbf{n} \in \mathbb{Z}^d, \tag{4.4}$$

where $\circ$ is the Hadamard product. As it is plausible to normalize the inputs to their bounds, we assume that each variable's period is 1. The Fourier series expansion of function (4.4) with $\mathbf{p} = \mathbf{1}^d$ is defined by Osgood (2019):

$$f(\mathbf{x}) = \sum_{\mathbf{n} \in \mathbb{Z}^d} c_{\mathbf{n}} e^{2\pi i \mathbf{n} \cdot \mathbf{x}}, \tag{4.5}$$

where $c_{\mathbf{n}}$ are the Fourier series coefficients, and they are calculated by:

$$c_{\mathbf{n}} = \int_{[0,1]^d} f(\mathbf{x}) e^{-2\pi i \mathbf{n} \mathbf{x}} d\mathbf{x}. \tag{4.6}$$

For real-valued functions, it holds that $c_{\mathbf{n}} = c^*_{-\mathbf{n}}$ where $c^*_{\mathbf{n}}$ is the conjugate of $c_{\mathbf{n}}$. We want to find the sin-cos form for the general case of dimension $d \in \mathbb{N}$. We use the concept of mathematical induction for this task. Therefore we show, that the equation is true for $d = 1$ and additionally prove, that if the equation holds for dimension $d - 1$ it is also valid for dimension d.

<u>$d = 1$:</u>

$$f(x) = \sum_{n \in \mathbb{Z}^1} c_n e^{2\pi i n \cdot x} \tag{4.7}$$

$$= \sum_{n \in \mathbb{N}} c_n e^{2\pi i n \cdot x} + \sum_{n \in \mathbb{N}} c_{-n} e^{-2\pi i n \cdot x} + c_0 \tag{4.8}$$

$$\begin{aligned} \overset{c^*_n = c_{-n}}{=} & \sum_{n \in \mathbb{N}} \big(\mathrm{Re}(c_n) + i\mathrm{Im}(c_n)\big)\big(\cos(2\pi n x) + i \sin(2\pi n x)\big) \\ & + \sum_{n \in \mathbb{N}} \big(\mathrm{Re}(c_n) - i\mathrm{Im}(c_n)\big)\big(\cos(2\pi n x) - i \sin(2\pi n x)\big) + c_0 \end{aligned} \tag{4.9}$$

$$= \sum_{n\in\mathbb{N}} 2\mathrm{Re}(c_n)\cos(2\pi nx) - 2\mathrm{Im}(c_n)\sin(2\pi nx) + c_0 \tag{4.10}$$

$$= \sum_{n\in\mathbb{N}_0} a_n\cos(2\pi nx) + b_n\sin(2\pi nx), \tag{4.11}$$

where

$$a_0 = c_0,\ a_n = 2\mathrm{Re}(c_n),\ b_n = -2\mathrm{Im}(c_n). \tag{4.12}$$

Assumption of the induction:
We will assume that the equation holds for $d-1$, where $d \geq 2$.

Induction step: $d-1 \to d$:
As the fourier series of any periodic and continous function is absolutely convergent, we are allowed to rearrange the sum in $(*)$ and receive

$$\sum_{\mathbf{n}=(n_1,\dots,n_d)\in\mathbb{Z}^d} c_{\mathbf{n}} e^{2\pi i\mathbf{n}\cdot\mathbf{x}} \tag{4.13}$$

$$\begin{aligned}
&\overset{(*)}{=} \sum_{n_1\in\mathbb{N}} \sum_{(n_2,\dots,n_d)\in\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i\mathbf{n}\cdot\mathbf{x}} \\
&+ \sum_{n_1\in\mathbb{N}} \sum_{(n_2,\dots,n_d)\in\mathbb{Z}^{d-1}} c_{-\mathbf{n}} e^{-2\pi i\mathbf{n}\cdot\mathbf{x}} \\
&+ \sum_{n_1=0}^{0} \sum_{(n_2,\dots,n_d)\in\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i\mathbf{n}\cdot\mathbf{x}}
\end{aligned} \tag{4.14}$$

$$\begin{aligned}
&\overset{c_{\mathbf{n}}^* = c_{-\mathbf{n}}}{=} \sum_{\mathbf{n}\in\mathbb{N}\times\mathbb{Z}^{d-1}} 2\mathrm{Re}(c_{\mathbf{n}})\cos(2\pi\mathbf{n}\cdot\mathbf{x}) - 2\mathrm{Im}(c_{\mathbf{n}})\sin(2\pi\mathbf{n}\cdot\mathbf{x}) \\
&\quad + \sum_{\mathbf{n}\in\{0\}\times\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i\mathbf{n}\cdot\mathbf{x}}
\end{aligned} \tag{4.15}$$

$$\begin{aligned}
&\overset{\text{Ind. asm.}}{=} \sum_{\mathbf{n}\in\mathbb{N}\times\mathbb{Z}^{d-1}} 2\mathrm{Re}(c_{\mathbf{n}})\cos(2\pi\mathbf{n}\cdot\mathbf{x}) - 2\mathrm{Im}(c_{\mathbf{n}})\sin(2\pi\mathbf{n}\cdot\mathbf{x}) \\
&\quad + \sum_{\mathbf{n}\in\{0\}\times\mathbb{N}_0\times\mathbb{Z}^{d-2}} a'_{\mathbf{n}}\cos(2\pi\mathbf{n}\cdot\mathbf{x}) + b'_{\mathbf{n}}\sin(2\pi\mathbf{n}\cdot\mathbf{x}),
\end{aligned} \tag{4.16}$$

where

$$\begin{aligned} a'_{\mathbf{0}} &= c_{\mathbf{0}}, \\ a'_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{3,\dots,d\} : n_2 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2\mathrm{Re}(c_{\mathbf{n}}) & \text{otherwise,} \end{cases} \\ b'_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{3,\dots,d\} : n_2 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2\mathrm{Im}(c_{\mathbf{n}}) & \text{otherwise.} \end{cases} \end{aligned} \tag{4.17}$$

Combining these two summands we get

$$f(\mathbf{x}) = \sum_{\mathbf{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1}} a_{\mathbf{n}} \cos(2\pi \mathbf{n} \cdot \mathbf{x}) + b_{\mathbf{n}} \sin(2\pi \mathbf{n} \cdot \mathbf{x}) \tag{4.18}$$

where

$$\begin{aligned} a_{\mathbf{0}} &= c_{\mathbf{0}}, \\ a_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2\mathrm{Re}(c_{\mathbf{n}}) & \text{otherwise,} \end{cases} \\ b_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2\mathrm{Im}(c_{\mathbf{n}}) & \text{otherwise.} \end{cases} \end{aligned} \tag{4.19}$$

And if we write equation (4.18) in vector form, we get

$$f(\mathbf{x}) = (a_{\mathbf{B}}, b_{\mathbf{B}}) \cdot \begin{pmatrix} cos(2\pi \mathbf{B} \cdot \mathbf{x}) \\ sin(2\pi \mathbf{B} \cdot \mathbf{x}) \end{pmatrix}, \tag{4.20}$$

where $a_{\mathbf{B}} = (a_{\mathbf{n}})_{\mathbf{n} \in \mathbf{B}}$, and $b_{\mathbf{B}} = (b_{\mathbf{n}})_{\mathbf{n} \in \mathbf{B}}$. Now, if we compare 4.2 and 4.20, we find similarities. We see that $(a_{\mathbf{B}}, b_{\mathbf{B}})$ is equivalent to $\mathbf{W}$, $\mathbf{b}$ is zero, and $\mathbf{B} = \mathbb{N}_0 \times \mathbb{Z}^{d-1}$ is the concatenation of all possible permutations of $\mathbf{n}$. As a result the perceptron represents a Fourier series. For practicality we limit $\mathbf{B}$ to

$$\mathbf{B} = \{0,\dots,N\} \times \{-N,\dots,N\}^{d-1} \setminus \mathbf{H}, \tag{4.21}$$

where N will be called the frequency of the mapping, $\mathbf{H} = \{\mathbf{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1} \mid \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}$. To find the number of element in the mapping, we will do the following calculations. We use $|\cdot|$ to talk about the number of elements in a set. Furthermore, we use the notation $[\![n]\!] := \{0,\dots,n\}$ for $n \in \mathbb{N}$ and $[\![m,l]\!] := \{m,\dots,l\}$ for

$m, l \in \mathbb{Z}$ and $m < l$. We have

$$\mathbf{B} = \{0,\dots,N\} \times \{-N,\dots,N\}^{d-1} \setminus \{\mathbf{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1} : \\ \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}.$$

It is immediately clear, that

$$|\{0,\dots,N\} \times \{-N,\dots,N\}^{d-1}| = (N+1)(2N+1)^{d-1},$$

therefore the only thing we need to show is the number of elements of $\mathbf{H}$. We will find it with mathematical induction. We start with $d = 2$:

$$\begin{aligned} &|\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!] : \exists j \in \{2\} : n_1 = 0 \wedge n_j < 0\}| \\ &= |\{\mathbf{n} \in \{0\} \times [\![-N,-1]\!]\}| \\ &= N \end{aligned}$$

Assumption of the induction:
We will assume that the equation holds for some d, where $d \geq 2$.

Induction step: $d \to d+1$:

$$\begin{aligned} &|\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^d : \exists j \in [\![2,d+1]\!] : \quad n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| \\ &= |\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^d : \exists j \in [\![3,d+1]\!] : \quad n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| + \\ &\quad |\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^d : \exists j \in \{2\} : \quad n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| \\ &= |\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^d : \exists j \in [\![3,d+1]\!] : \quad n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| + \\ &\quad |\{\mathbf{n} \in \{0\} \times [\![-N,-1]\!] \times [\![-N,N]\!]^{d-1}\}| \\ &= |\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^{d-1} : \exists j \in [\![2,d]\!] : \quad n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| + \\ &\quad N(2N+1)^{d-1} \\ &\overset{\text{Ind. asm.}}{=} \sum_{l=0}^{d-2} N(2N+1)^l + N(2N+1)^{d-1} = \sum_{l=0}^{d-1} N(2N+1)^l. \end{aligned}$$

Hence, we calculate the dimension m of all possible permutations.

$$m = (N+1)(2N+1)^{d-1} - \sum_{l=0}^{d-2} N(2N+1)^l. \tag{4.22}$$

We can find the Fourier series coefficients by sampling the function uniformly with a frequency higher than the Nyquist frequency and applying a Fast Fourier Transform (FFT) on the sampled signal. The resulting FFT coefficients are the Fourier series coefficients multiplied by the number of the sampled points. Furthermore, in theory, if we initial-

ize the weights with the Fourier series coefficients, our network should give the training target at iteration 0. Moreover, we take advantage of the Fourier series properties, for example, convergence proofs.

4.3.2 SIRENs and Fourier mapping comparison

In this section we want show that a Fourier mapped perceptron is structurally like a SIREN with one hidden layer. If we evaluate $\mathbf{W} \cdot \gamma(\mathbf{x})$ in equation (4.2), using (4.3) , we get:

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W} \cdot \begin{pmatrix} cos(2\pi \mathbf{B} \cdot \mathbf{x}) \\ sin(2\pi \mathbf{B} \cdot \mathbf{x}) \end{pmatrix} + \mathbf{b}.$$

If we set $\mathbf{B} = (\mathbf{B}_1, \ldots, \mathbf{B}_m)^T$, with $\mathbf{B}_i \in \mathbb{R}^{1 \times d}$, then the first summand is equal to

$$\begin{pmatrix} \sum_{k=1}^{m} W_{1,k} c(2\pi \mathbf{B}_k x) + \sum_{k=1}^{m} W_{1,m+k} s(2\pi \mathbf{B}_k x) \\ \vdots \\ \sum_{k=1}^{m} W_{d_o,k} c(2\pi \mathbf{B}_k x) + \sum_{k=1}^{m} W_{d_o,m+k} s(2\pi \mathbf{B}_k x) \end{pmatrix}^T$$
$$= \begin{pmatrix} \sum_{k=1}^{m} W_{1,k} s(2\pi \mathbf{B}_k x - \pi/2) + \sum_{k=1}^{m} W_{1,m+k} s(2\pi \mathbf{B}_k x) \\ \vdots \\ \sum_{k=1}^{m} W_{d_o,k} s(2\pi \mathbf{B}_k x - \pi/2) + \sum_{k=1}^{m} W_{d_o,m+k} s(2\pi \mathbf{B}_k x), \end{pmatrix}^T$$

where s and c are short forms of sine and cosine. And if we define $\phi = (-\pi/2, \ldots, -\pi/2, 0, \ldots, 0)^T \in \mathbb{R}^{2m}$ and $\mathbf{C} := (\mathbf{B}, \mathbf{B})^T$, we result in

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W} \cdot sin(2\pi \mathbf{C} \cdot \mathbf{x} + \phi)^T + \mathbf{b}.$$

Here we see that $\mathbf{C}$ is acting as the weight matrix applied to the input, ϕ is like the first bias vector and $sin(\cdot)$ is the activation function. Hence, the initial Fourier mapping can be represented by an extra initial SIREN layer, with the difference that $\mathbf{B}$ and ϕ are trainable in the SIREN case. This finding closes the bridge between Fourier frequency mappings and sinusoidal activation functions, which have attracted much attention.

4.3.3 Progressive training

Lin *et al.* (2021) introduced a training strategy for coarse-to-fine registration for NeRFs which they called Bundle-Adjusting Neural Radiance Fields (BARF). Their idea is to mask out the positional encoding's high-frequency activations at the start of training and gradually allow them during training. Their work showed how to use this strategy on positional encodings to improve camera registration. Our work will show how to run this strategy on an arbitrary Fourier mapping and show that it improves the generalization of

the interpolation task. We weigh the frequencies of γ as follows:

$$\gamma^{\alpha}(\mathbf{x}) := \begin{pmatrix} w^{\alpha}_{\mathbf{B}} \\ w^{\alpha}_{\mathbf{B}} \end{pmatrix} \circ \gamma(\mathbf{x}) \tag{4.23}$$

where $w^{\alpha}_{\mathbf{B}}$ is the element wise application of the function $w_{\alpha}(z)$ on the vector of norms of $\mathbf{B}$ on the input dimension:

$$w^{\alpha}_{\mathbf{B}} := w_{\alpha} \begin{pmatrix} ||B_1||_2 \\ \vdots \\ ||B_m||_2 \end{pmatrix}. \tag{4.24}$$

where $w_{\alpha}(z)$ is defined as:

$$w_{\alpha}(z) = \begin{cases} 0 & \text{if } \alpha - z < 0 \\ \frac{1-\cos((\alpha-z)\pi)}{2} & \text{if } 0 \leq \alpha - z \leq 1 \\ 1 & \text{if } \alpha - z > 1 \end{cases} \tag{4.25}$$

Here, $\alpha \in [0, \max((||B_i||_{d_{\text{in}}})_{i \in \{1,\dots,m\}})]$ is a parameter which is linearly increased during training. This strategy forces the network to train the low frequencies at the start of training, ensuring that the network will produce smooth outputs. Later, when high-frequency activations are allowed, the low-frequency components are trained, and the network can focus on the left details. This strategy should reduce the effect of overfitting, which was introduced by Tancik *et al.* (2020) when using mappings with large standard deviations.

4.3.4 Pruning

The standard way of using equation (4.21) is by defining a value N and taking the whole set $\mathbf{B}_N$. High-dimensional tasks lead to high memory consumption, and it is not clear whether this subset of $\mathbb{Z}^d$ brings the best performance. We, therefore, propose a way to select a more appropriate subset through data pruning. A pruning $pr(N, M)$ is done as follows: Assume we have $N, M \in \mathbb{N}$ with $M >> N$ and $|\mathbf{B}_N| = n$, $|\mathbf{B}_M| = m$. We train a perceptron with an integer mapping given by $\mathbf{B}_M$. After training we define $\mathbf{D}$ such that $\mathbf{D}$ contains only those elements of $\mathbf{B}_M$ where the respective weights are greater than a margin, that is chosen to yield $|\mathbf{D}| = n$. While $\mathbf{B}_N$ and $\mathbf{D}$ now have the same size, we believe that $\mathbf{D}$ will yield better performance because it contains the most relevant frequencies of the signal we want to reconstruct.

4.3.5 Integer lattice mapping applied to MLPs

Although we showed in section 4.3.1 that we could represent any bounded input function with only one Fourier mapped perceptron, in practice, these networks can become very

wide to give high performance. As a result, the number of calculations will increase. To compromise between performance and speed, one can add depth and reduce the width of the network.

First, it is natural that using MLPs rather than perceptrons increases the performance. However, it remains unclear why our proposed integer mapping should perform better than competing mappings for multilayer networks. One could argue that if a mapping gives the perceptron a high representation power, it will also provide a high representation power to the MLP and vice versa. First, however, we should verify this claim with experiments.

In addition, we remind the reader that a periodic function has integer frequencies. Moreover, because our assumption that the signal we want to reconstruct is periodic, it will have only integer frequencies. Also, the activation functions we are using only introduce integer frequencies when applied to a periodic function, as we will see in the next section. With this, we reduce the search space for frequencies from $\mathbb{R}$ to $\mathbb{Z}$, which could make the optimization easier as the search space is more compact and approachable.

4.3.6 Activation functions on periodic signals

We claim that when we apply an integer mapping to the input, we force the network output to be periodic. This comes from the fact that the frequencies introduced by the activations are integers, and a periodic signal has only integer frequencies. To prove this claim, we will first analyze the frequencies in the 1D case and later demonstrate those findings in 2D experiments. As an initial Fourier mapping involves using a sinus function on the mapped input, we now discuss the effect of applying an activation function on top of a sinus representation. Applying a ReLU or sine on a mapped input will produce frequencies that are multiples of its input frequencies. For example, if we apply a ReLU to a sine function, we get

$$ReLU(\sin(x)) = \frac{1}{\pi} + \frac{\sin(x)}{2} + \sum_{\substack{n=2k \\ k\in\mathbb{N}}} \frac{2}{\pi(1-n^2)} \cos(nx), \tag{4.26}$$

and if we apply a sine to a sine we get

$$\sin(A \cdot \sin(x)) = 2 \sum_{n=0}^{\infty} J_{2n+1}(A) \sin((2n+1)x), \tag{4.27}$$

where J_i are Bessel functions. In these cases, we can immediately see that the output frequencies are multiples of the input frequencies. Motivated by these findings, we explore whether it does generalize to higher dimensional signals.

Finding an analytical solution for the d-dimensional case is complicated, so we decided to show it empirically. We define $\mathbf{B}$ in two different ways. First we generate $\mathbf{B}_N$ limited by $N = 2$, responsible for the integer mapping, for the Gauss mapping we

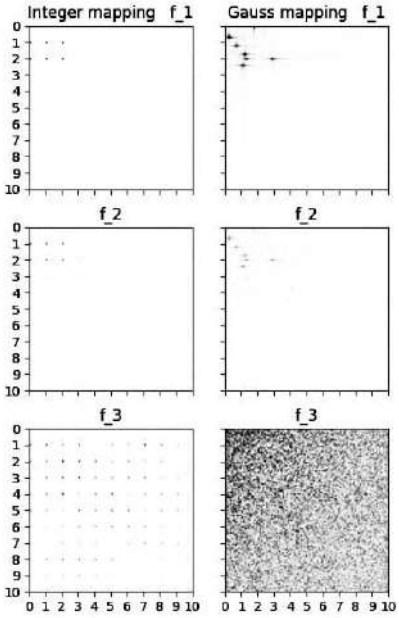


Figure 4.2: The effect of the common activation functions on the spectrums of functions with integer and non-integer frequencies.

sample $\mathbf{B}$ from a Gaussian distribution with mean, variance and dimension according to the previous $\mathbf{B}_N$, to achieve maximal comparability. We then compare the spectrum of $f_1 := \gamma(x) \cdot \mathbf{1}$, $f_2 := \text{ReLu}(\gamma(x) \cdot \mathbf{1})$ and $f_3 := \sin(\gamma(x) \cdot \mathbf{1})$, where $\mathbf{1}$ represents the weight matrix, in this example defined to only contain 1's. We visualize the spectrum of our results in Fig. 4.2 in the range of $(0,10)^2$.

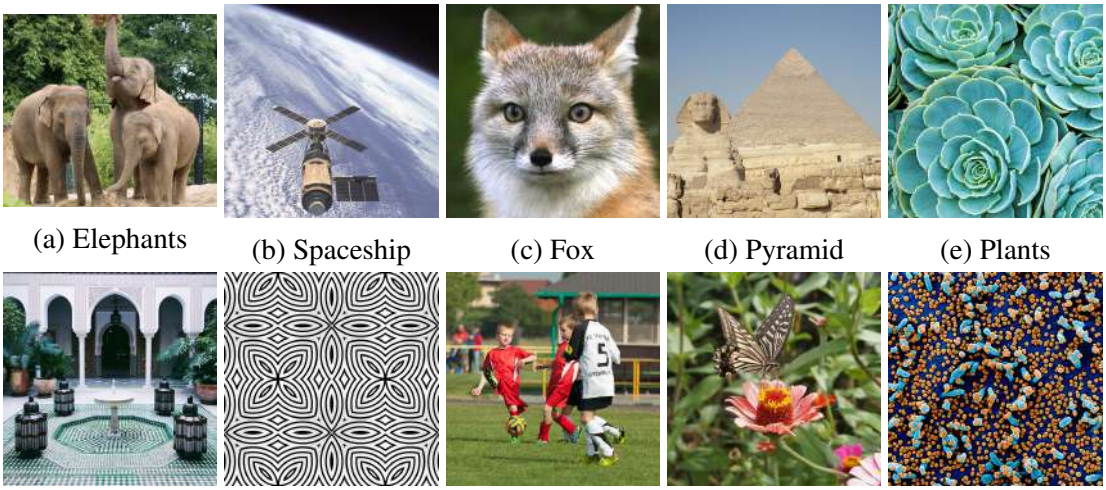
When we apply a non-linearity to the function with integer frequencies, the output spectrum has only integer frequencies, which means that it is periodic. Also mentionable is the beautiful alignment of the high frequencies, which contrasts the Gauss mapping, where no clear pattern is present. Moreover, we see that the sine activation produces more high-frequency components than ReLUs, which could explain why sine activations are more effective in shallow networks.

4.4 Experiments

4.4.1 Weight initialization and progressive training

In this section, we want to prove our mathematical claims through experiments. First, we will show that the derivation of the integer mapping indeed represents the Fourier series. Secondly, we want to check whether progressive training helps with generalization.

We conducted our experiments on the image regression task. This task aims to make



(a) Elephants (b) Spaceship (c) Fox (d) Pyramid (e) Plants

(f) Islamic arch (g) Pattern (h) Football (i) Butterfly (j) COVID-19

Figure 4.3: The images used in the image regression experiments.

a neural network memorize an image by predicting the color at each pixel location. We use ten images with a resolution of 512×512, which we see in figure 4.3, and report the mean peak signal-to-noise ratio (PSNR). We divide the image into train and test sets, where we use every second pixel for training and take the complete image for testing. We utilize 3 Fourier-mapped perceptrons with $N = 128$ (Nyquist frequency), one for each image channel. We normalize the input ($\mathbf{x}$) to have an interval between [0,1] in both width (x) and height (y) dimensions.

(a) PT=F, WI=F (b) PT=F, WI=T (c) PT=T, WI=F (d) PT=T, WI=T (e) GT

Figure 4.4: A visualization of the outputs of Fourier mapped perceptrons of $N = 128$. PT stands for progressive training and WI stands for weight initialization. T/F stands for True/False, respectively.

In this experiment, we made an ablation study: With and without weight initialization using the normalized FFT coefficients of the image's training pixels, with and without the progressive training scheme explained in section 4.3.3. For progressive training, α was linearly increased from 0 to its maximum value at 75% of training iterations. In training, we only make an update step after we accumulate the gradients of the whole image. We did not study learning schedules in this work, and the reader is encouraged to

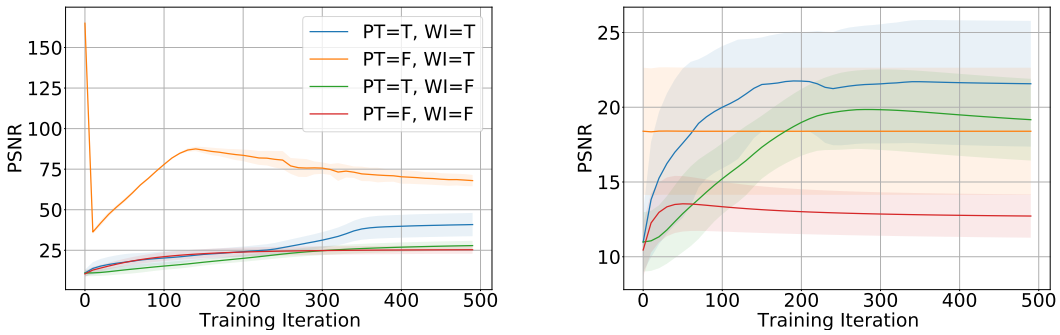


Figure 4.5: The training progress of Fourier mapped perceptrons with $N = 128$. The left and the right figures report the train and test PSNR, respectively. Note: The y-axis limits are different in both plots. WI without PT yields a PSNR of 160, which one can consider as the ground truth proving that the perceptron is a Fourier series.

try different schedules. Figure 4.4 shows a visualization of one of the images, and figure 4.5 shows the training progress, where the solid line is the mean PSNR and the shaded area shows the standard deviation.

As can be deducted from figure 4.5, one can see that the training PSNR starts at an optimum at the start of training when we use weight initialization (WI), and we do not use progressive training (PT). This fact underlines our claim that *a perceptron with an integer lattice mapping is indeed a Fourier series*. Note that in case both WI and PT are used, the training PSNR is not optimal at the start, because the PT masks out high-frequency activations.

We can also see from figure 4.5 that whenever we use progressive training, it always shows a higher test PSNR, which certifies that *progressive training helps with generalization*. Lastly, the perceptron overfits the training pixels when we did not employ PT and WI. This overfitting can be seen quantitatively with a very low test PSNR (red line in figure 4.5) and qualitatively with grid-like artifacts (Figure 4.4a)).

4.4.2 Perceptron experiments

In this experiment, we want to compare the representation power of the different mappings in the single perceptron case. We conducted our experiments in the same setting as in section 4.4.1, where we used progressive training and did not use weight initialization.

In the integer mapping, we increased N's value from 4 to half the training image dimension (Nyquist frequency) and calculated all possible permutations $\mathbf{B}_N$, as discussed in section 4.3.1. For the Gaussian mapping, we sample $m = |\mathbf{B}_N|$ parameters from a Gaussian distribution with a standard deviation of 10 (which was the best value for this task in our experiments). Also, we test a one-layer SIREN with one hidden layer having

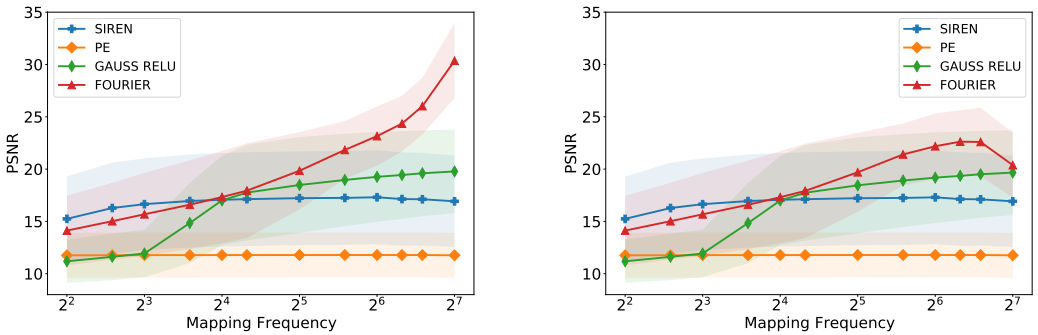
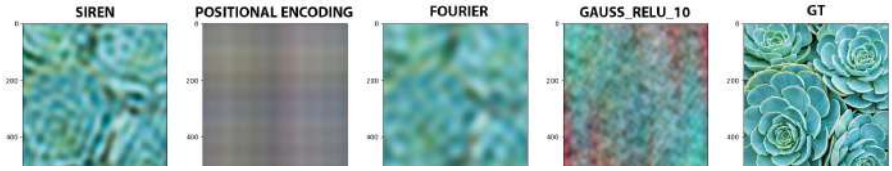


Figure 4.6: Perceptron experiments with different values for the mapping frequency N. We report the train PSNR on the left and the test PSNR on the right. For high values of N our integer mapping outperforms all competing mappings.

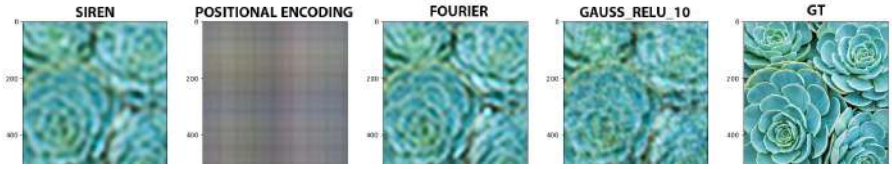
the same size m. Finally, we adopt the positional encoding (PE) scheme from Mildenhall *et al.* (2020) and limit its values to N. Figure 4.6 shows our experiments' results on the training and test pixels, respectively. Figure 4.7 shows the networks' outputs trained on one of the images.

At low N values (figure 4.7a), we see that the Gaussian mapped perceptrons do not work because the number of sampled frequencies is too low, so there is a low chance that samples will be near the image's critical frequencies. On the other hand, the integer mapped perceptrons give a blurry image because they can only learn low frequencies. The SIREN performs relatively well in this case, and we think this is because SIRENs naturally inherit a learnable Fourier mapping that is not restricted to the initial sampling, as described in section 4.3.2. PE can only produce horizontal and vertical lines because it has diagonal frequencies (only one non-zero frequency is allowed), and this effect is persistent at any value of N.

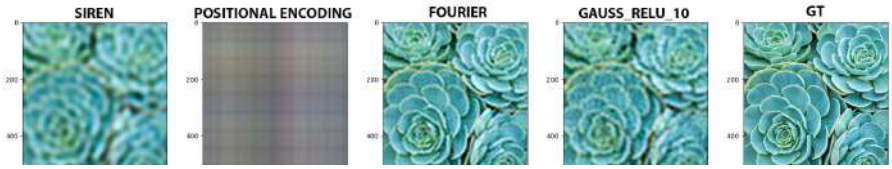
As N increases, SIREN, Gauss, and integer mapping performance increase, giving similar performance around $N = 16$ (figure 4.7b)). For high values of N, we see that in figure 4.7d), the integer lattice mapping of the Fourier coefficients outperforms the competing mappings, clearly displaying more details in the reconstruction. On the other hand, the PSNR of the SIREN and the Gaussian mapped perceptrons saturates. We think this is because both mappings rely on sampling the frequencies. Although we can get many of the critical frequencies of the image with sampling, it is improbable to get all of them simultaneously. Even the trainability of the SIREN mapping did not help in this case.



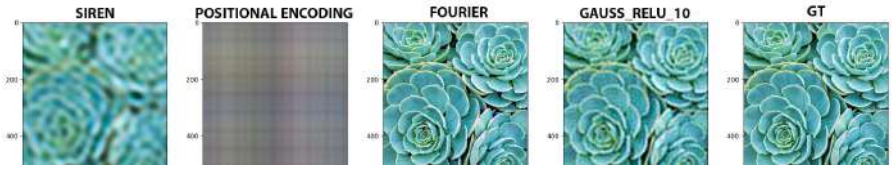
(a) Network predictions using $N = 8$



(b) Network predictions using $N = 16$



(c) Network predictions using $N = 32$



(d) Network predictions using $N = 128$

Figure 4.7: The visualization of the Fourier mapped perceptrons and the one layer SIREN with different values of N. GT stands for ground truth.

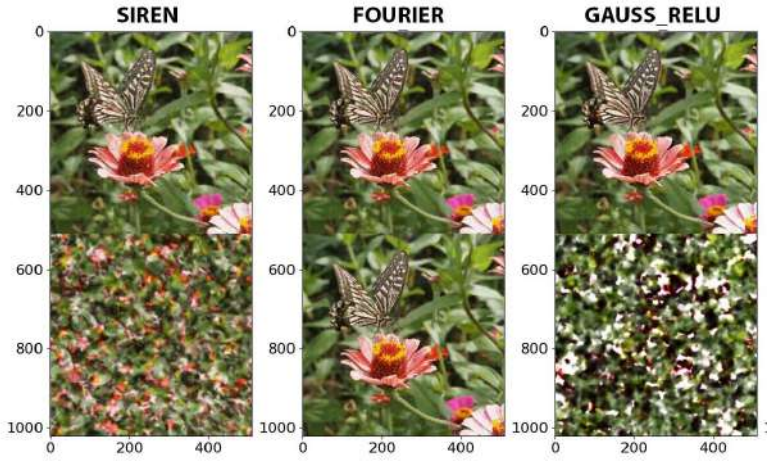


Figure 4.8: Visualization of SIREN (Sitzmann *et al.*, 2020), Gauss ReLU (Tancik *et al.*, 2020) and our method. The top row is the first period and bottom row is the second period, which shows our method enforcing periodicity.

4.4.3 MLP experiments

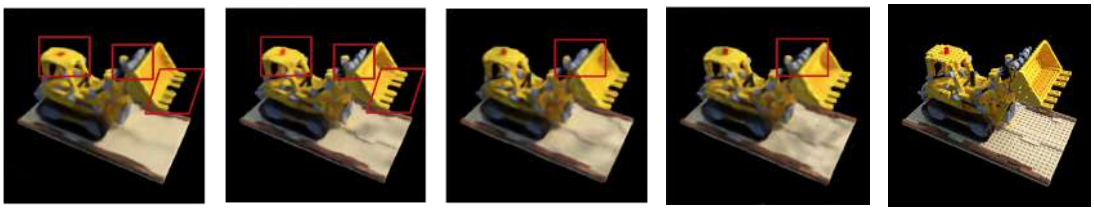
Our theory for integer mapping assumes an underlying function that is periodic. However, it is not clear that we will end up with a periodic function if we use an integer mapping. In this experiment, we want to check if applying an integer mapping forces periodicity. Secondly, we want to validate our claim (in section 4.3.5) that if a mapping gives the perceptron a high representation power, it will also give a high representation power to the MLP and vice versa. We compared ReLU networks with integer, Gaussian, PE, and pruned integer mapping (section 4.3.4). We also compared SIRENS with no mapping (extra layer), integer, pruned mapping. We made a grid search of the parameters $N = [8, 16, 32]$, depth= $[0, 2, 4, 6]$ (depth=0 represents a perceptron), and fixed the width to 32. We used a $pr(N, 128)$ for the pruned mapping.

Activ.	Map.	N=8 m=113				N=16 m=481				N=32 m=1985			
		d=0	d=2	d=4	d=6	d=0	d=2	d=4	d=6	d=0	d=2	d=4	d=6
Sine	No	**16.65**	22.15	**23.26**	**24.07**	17.07	22.09	23.84	19.76	17.22	14.90	14.67	13.63
	Int.	15.68	**22.31**	22.41	20.94	**17.33**	27.66	27.06	**27.33**	**19.84**	33.78	26.98	23.60
	Pr.	15.28	21.03	22.40	23.00	16.76	**28.17**	**27.68**	24.66	18.48	37.34	30.41	19.74
Relu	P.E.	11.78	16.61	17.37	17.77	11.78	16.87	17.79	17.95	11.78	17.05	18.15	18.15
	Gs. σ_{10}	11.93	21.90	21.68	21.69	17.01	24.53	24.26	25.13	18.48	26.10	26.30	27.48
	Gs. σ_{pr}	14.06	20.23	20.78	20.88	12.69	26.02	26.40	26.72	13.01	37.69	**37.90**	**37.74**
	Int.	15.68	20.51	20.65	20.62	**17.33**	24.42	24.09	24.49	**19.84**	31.57	32.14	32.79
	Pr.	15.28	20.35	20.92	20.96	16.76	25.87	26.23	26.33	18.48	**37.70**	36.81	37.48

Table 4.1: The mean training PSNR results of network type comparison experiment with varying network depth (d), number of frequencies (N). We use the following abbreviations: Activ. = Activation function, Map. = Mapping type, Int. = Integer, Pr.= Pruned Integer, P.E. = Positional Encoding, Gs. = Gaussian. Here, m is the mapping size and σ is the standard deviation.

Activ.	Map.	N=8 m=113				N=16 m=481				N=32 m=1985			
		d=0	d=2	d=4	d=6	d=0	d=2	d=4	d=6	d=0	d=2	d=4	d=6
Sine	No	**16.65**	21.63	**21.85**	**21.99**	17.06	21.28	22.03	18.50	17.22	13.57	13.29	12.37
	Int.	15.68	**21.75**	21.53	20.06	**17.31**	**23.48**	**22.67**	22.28	**19.70**	16.85	17.89	16.36
	Pr.	15.28	20.49	21.22	21.45	16.75	22.00	21.39	22.17	18.39	20.49	15.15	13.13
Relu	P.E.	11.78	16.60	17.33	17.70	11.78	16.85	17.73	17.87	11.79	17.02	18.06	18.02
	Gs. σ_{10}	11.93	20.67	21.06	20.90	17.00	22.96	22.78	23.04	18.45	23.66	23.61	**23.73**
	Gs. σ_{pr}	14.06	19.89	20.22	20.21	12.69	22.46	22.48	22.16	12.99	23.12	23.48	23.33
	Int.	15.68	20.27	20.35	20.23	**17.31**	22.93	22.65	**22.50**	**19.70**	**24.36**	**24.02**	**23.73**
	Pr.	15.28	19.98	20.33	20.21	16.75	22.31	22.26	22.09	18.39	23.24	23.18	23.30

Table 4.2: The mean test PSNR results of network type comparison experiment. For abbreviations see table 4.1.



(a) ReLu w/o pr. (b) ReLu w/ pr. (c) Sine w/o pr. (d) Sine w/ pr. (e) GT

Figure 4.9: View synthesis results using a simplified Nerf. A small MLP with a depth of 4, width of 64 and integer mapping with a frequency of 4 is used. The pruning is done with pr(4,8). The pruning technique shows qualitative improvements. ”pr” is a shortcut for pruning.

For the Gaussian mapping, we had two settings. The first one had a standard deviation of 10 (σ_{10}), which had the best performance in the perceptron experiments. In the second one, we set the standard deviation the same as the pruned integer mapping's standard deviation (σ_{pr}) to check its effect. Tables 4.1 and 4.2 show the mean train and test PSNRs respectively.

Figure 4.8 shows a visualization of the network's outputs at $N = 16$, depth = 4 and width = 32 for the first period and next period in the height and width directions ($f([x+1, y+1])$). And we see that *the integer mapping forces the network's underlying function to be periodic* unlike others, which proves our first hypothesis.

From the table 4.1 we see that if a mapping at $d = 0$ gives the highest PSNR, this does not mean that it will give the highest PSNR for $d > 0$ and vice versa. One clear example at $N = 32$ is the Gauss σ_{pr}, where it has a PSNR of 13.01 dB at $d = 0$, which is lower than integer mapping (19.84 dB), but has the highest PSNR at $d = [4, 6]$. This result disproves our initial assumption that if a mapping gives the perceptron a high representation power, it will also give a high representation power to the MLP. We see also that the pruned integer mapping has comparable results with the Gauss σ_{pr}, and this shows that the main contributor to the performance is the mappings' standard deviation.

From the tables, we can also observe some trends. First, networks with sine activations and large mappings collapse during training. and perform worse than Relu networks. Second, the integer mapping usually gives the best test PSNR, demonstrating its effectiveness in the MLP case. Third, the pruned integer mapping shows consistently better train PSNR than the normal integer mapping at $d > 0$. We believe this is because pruned mapping has a higher standard deviation. Finally, the PE is worse in every case because we cannot easily control the standard deviation, and it has very few parameters.

4.4.4 Novel view synthesis experiments

This section wants to see if our findings in the image regression task transfer to the novel view synthesis (NVS) task. In NVS, we use 2D images of a scene to find its 3D representation. With this representation, one can render images from new viewpoints. In

Act.	Map.	$N = 4$				N=8
		d=0	d=2	d=4	d=6	d=0
Sine	No	**20.37**	23.08	**23.55**	23.35	OM
	Int.	18.42	22.22	22.95	22.97	**19.31**
	Pr.	19.15	**23.12**	**23.58**	23.36	-
Relu	P.E.	16.30	21.48	22.64	23.51	16.40
	Gs.	18.93	22.81	**23.64**	**23.82**	19.29
	Int.	18.42	21.81	22.68	23.28	**19.31**
	Pr.	19.15	22.78	23.61	**23.89**	**-**

Table 4.3: Validation PSNR scores of NVS experiments using a mapping of frequency 4. OM stands for out of memory. For other abbreviations see table 4.1.

contrast to the 2D experiments, we map the (x, y, z) input coordinates to a 4-dimensional output, the RGB values, and a volume density. We use a simplified version of the official Neural radiance fields (NeRF) (Mildenhall *et al.*, 2020) for this experiment, where we remove the view dependency and hierarchical sampling. Here, we experiment with the input mappings used in section 4.4.3. Unless otherwise stated, we adopt the settings from the image regression task. We set the network width to be 64.

As the mapping size increases exponentially, we do our experiments with lower frequencies than in the 2D case. Specifically, we used the integer mapping on four frequencies. The frequencies of our mapping were limited to the maximum network size which we could fit on NVIDIA GTX-2080Ti. The pruning is given by $pr(4, 8)$. We conducted our experiments on the bulldozer scene, commonly used for Nerf experiments. For training, we used a batch size of 128, 50.000 epochs, and a learning rate of 5×10^{-4}.

As seen in Table 4.3, in the perceptron case ($d = 0$), SIREN provides the best performance, which aligns with our image regression results at low values of N. We observe that the pruned mapping increases the performance compared to the normal mapping for both Relu and sinusoidal activation. This increase in performance is because the pruned mapping has a higher standard deviation than the normal mapping. We see qualitative improvements in the pruning experiment in figure 4.9. Gauss gives comparable results to pruned integer mapping because they have the same standard deviation. These findings align with our conclusions from image regression experiments. However, we could not test a perceptron with frequencies higher than 8, which was superior in image regression, due to memory limitations.

4.5 Conclusion

This work showed that a Fourier mapped perceptron with an integer lattice mapping is precisely the d-dimensional Fourier series. As a result, one perceptron with a large enough lattice can represent any signal. We demonstrated experimentally on the im-

age regression task that one perceptron with frequencies equal to the Nyquist rate of the whole image could reconstruct it perfectly. Furthermore, we showed that our modified progressive training strategy of adding sequentially low to high frequencies worked on arbitrary mappings and improved the generalization of the interpolation task. In addition, we showed that a Fourier mapped perceptron is structurally like a one hidden layer SIREN but with a trainable mapping. We saw that trainable mappings help if we train with a low number of frequencies and hurt when we use a high number of frequencies. Furthermore, in Fourier mapped MLPs, we showed that the integer lattice mapping forces the neural network's underlying function periodicity. Lastly, we confirmed experimentally on the image regression and novel view synthesis tasks that the main contributor to reconstruction performance using a Fourier mapped MLP is the size of its mapping and the standard deviation of its elements.

Chapter 5

Instance segmentation using implicit neural representation

5.1 Motivation

In chapter 4, we showed the connection between INRs and Fourier series. Moreover, we showed the advantages of using INRs in systems over the traditional discrete representations. However, these INRs were inherently trained on single objects and have not yet been adopted for a general task, particularly the task of instance segmentation. In this chapter, we use INRs to represent masks in instance segmentation. In particular, we will use them to parameterize a level set function.

Unlike FourierNet (Riaz *et al.*, 2020), which could only represent instances that have single star-shaped masks, this representation can theoretically represent any mask. Moreover, like FourierNet, as the Fourier series is a special-case INR, the Fourier series? low-frequency components hold the general mask shape, and high-frequency components hold the edges of the mask. Therefore, our representation is also meaningful, and we can compress it according to the use case. Furthermore, as INRs are continuous in the domain of input coordinates, we can sub-sample the pixel coordinates to generate higher resolution masks during inference, which is a significant advantage compared to other mask representations.

Our work will focus on the accuracy aspect of instance segmentation rather than memory efficiency, because of these advantages. It will build on detect-then-segment approaches with Mask R-CNN (He *et al.*, 2017) as the baseline. We will show that our representation performs better than previously dominant grid-based representations. This chapter is based on our paper 'FourierMask: Instance Segmentation Using Fourier Mapping in Implicit Neural Networks' (Riaz *et al.*, 2021).

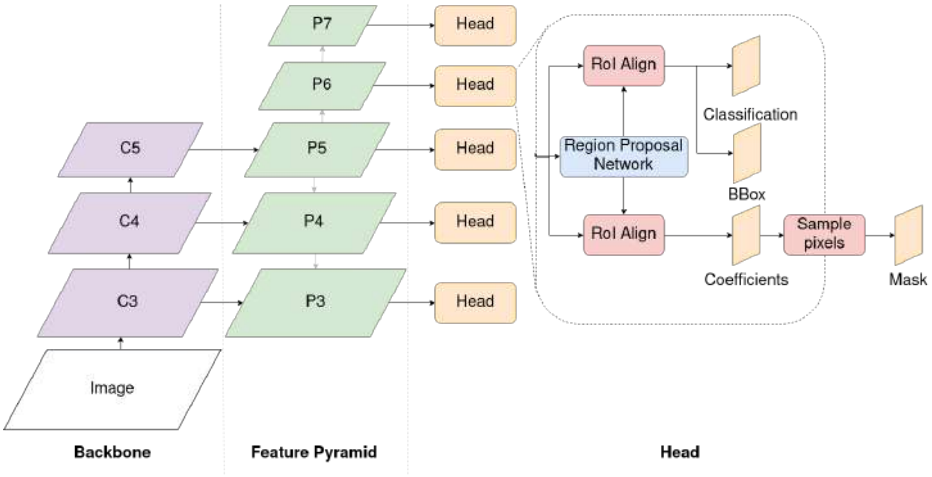


Figure 5.1: The overall architecture of FourierMask.

5.2 Method

5.2.1 Architecture

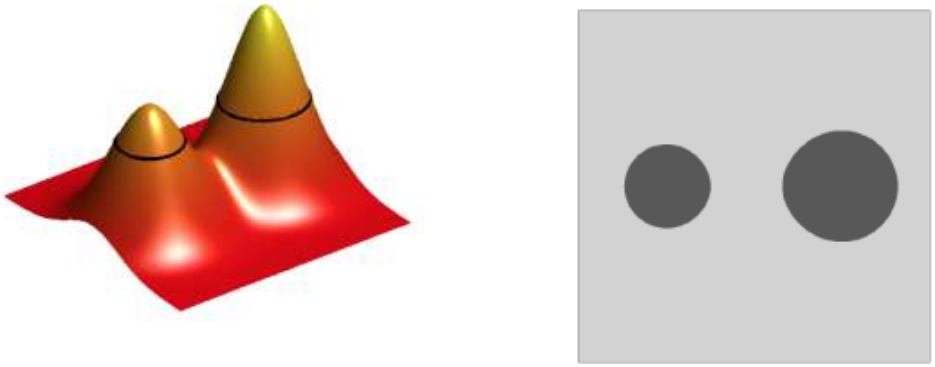
As in this work we focus on accuracy, we designed FourierMask to be a two-stage instance segmentation method. FourierMask builds on Mask R-CNN (He *et al.*, 2017), and figure 5.1 shows its architecture. It has a backbone followed by a feature pyramid network (FPN) (Lin *et al.*, 2017a). From all of the feature levels in FPN, we use region proposal networks (RPN) to generate proposal candidates. Next, we use an ROI align operation to generate fixed-size feature maps from all selected proposal candidates. Finally, we provide the feature maps to the network head, which has two branches. One branch predicts the proposal's class, box parameters, and the other predicts the function coefficients. We designed two network head designs, which we will discuss in the next section along with the mask representation.

5.2.2 Mask representation

We represent the mask as a super level set, as shown in figure 5.2. We consider a location in the mask as foreground if the level set function is higher than a threshold and background, if it is lower than the same threshold. We define the super level set as follows:

$$L_c^+(y) = \{\mathbf{x} | y(\mathbf{x}, \mathbf{W}) \geq c\} \tag{5.1}$$

Here $y(\mathbf{x}, \mathbf{W}) \in \mathbb{R}$ is the level set function, which we parameterize as a Fourier series or a coordinate-based MLP, $\mathbf{x} \in [0,1]^2$ is the normalized pixel coordinates vector (i, j) in the proposal candidate box, $\mathbf{W}$ are the parameters that control the level set function, and



(a) The level set function (b) The output mask

Figure 5.2: An illustration of how the mask is represented. The left image shows the level set function with the threshold in black, and the right image shows the resulting mask. We can see that it can represent multi mask instances, unlike FourierNet.

c is the level set threshold. In chapter 4, we proved that a Fourier mapped perceptron with an integer grid mapping is a Fourier series. So in the case we want to represent the level set as Fourier series, we use equation 4.2 to define the level set function:

$$y(x, \mathbf{W}) = \mathbf{W}.\gamma(\mathbf{x}) \tag{5.2}$$

where here $\mathbf{W} \in \mathbb{R}^{1\times 2m}$ are weights of the perceptron and at the same time the Fourier series coefficients, $\gamma(\mathbf{x})$ is the Fourier mapping defined as

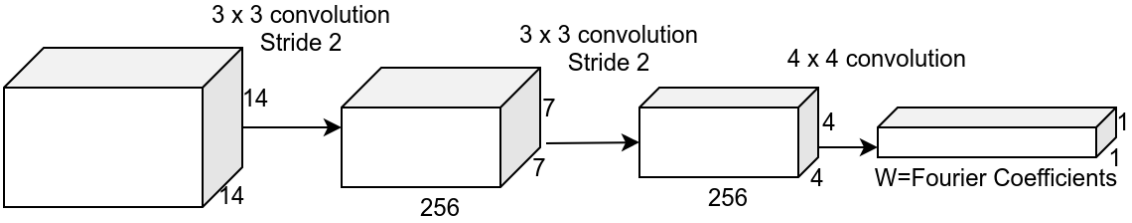
$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi\mathbf{B}\cdot\mathbf{x}) \\ \sin(2\pi\mathbf{B}\cdot\mathbf{x}) \end{pmatrix}. \tag{5.3}$$

Here $\mathbf{B} \in \mathbb{R}^{m\star 2}$ is the Fourier mapping matrix, and its elements are the integer grid elements as given in equation 4.21. We can calculate the number of elements m in the mapping with equation 4.22, where the number of dimensions is 2 ($d = 2$).

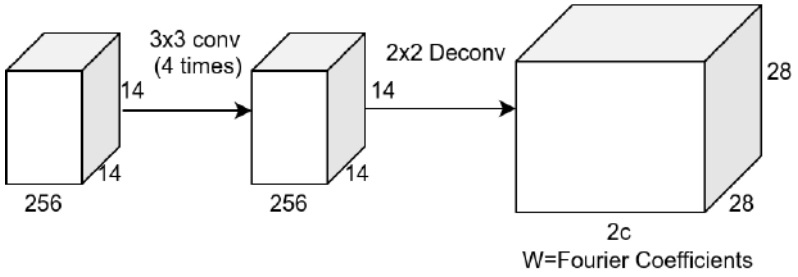
$$m = (N+1)(2N+1) - N \tag{5.4}$$

Here N is the number of frequencies used in the series. Compared to FourierNet, this representation can represent any mask, and it is not restricted to single star-shaped instance masks. The network head predicts the Fourier series coefficients $\mathbf{W}$ during inference, as shown in figure 5.3a). Then we evaluate the level set function (equation 5.2) at the pixel locations we desire. First, we evaluate pixels in a grid pattern, and then we further sub-sample in a more refined grid if we need a higher resolution mask.

Furthermore, we designed a second head where we divide the mask into sparse grid patches, and at each location, we predict the coefficients for that patch, as shown in



(a) Head design for predicting the mask coefficients for the whole image.



(b) Head design for predicting the mask coefficients for each location in the feature map.

Figure 5.3: The two FourierMask head architecture designs for a ROI align size of 14×14. The network predicts the Fouier coefficients $\mathbf{W}$ for a) the whole image, b) each location in the feature map.

figure 5.3b). The advantage of this design is that the underlying function to represent the patch is simpler than the whole mask, so learning the underlying patch function should be simpler. On the other hand, the memory requirement of the second design is much higher. Therefore, we should pick either design, depending on the system requirements. Finally, in case we want to represent it as a coordinate MLP, we pass $\gamma(\mathbf{x})$ to an MLP and repeat the same steps during inference. However, because the number of parameters in the MLP is significant, the network head predicts only the weights of the first layer.

5.2.3 Mask representation upper bound

In this section, we will test the representation power of FourierMask similar to what we did in FourierNet in section 3.2.3. First, we reconstruct the masks with the representation at different frequencies and then compare them to the ground truth masks of the data set. Along with checking the representation power, this experiment gives an insight into the optimal number of frequencies for a dataset. We performed this experiment by applying a 2D FFT on all the target object masks in the MS COCO training dataset. This Fourier transform gave us the coefficients of a Fourier series, which holds the same meaning as the prediction of the coefficients $\mathbf{W}$ of FourierMask. First, we experimented with only the lower frequency coefficients of the Fourier series and reconstructed the object's mask by applying equation 5.2. We did this for all the objects' masks in the MS COCO training set and evaluated the IoU loss of the reconstruction compared to the target. Then

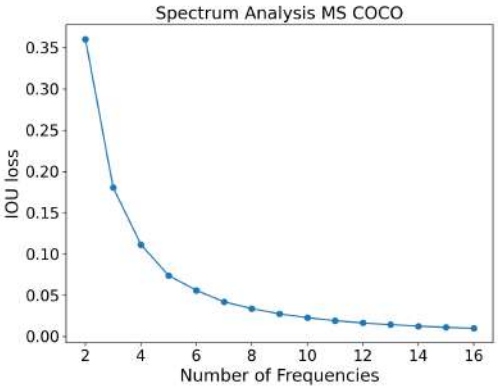
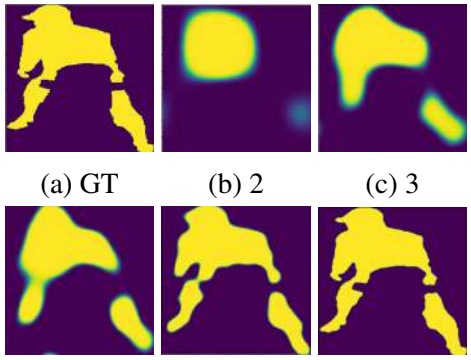


Figure 5.4: Spectrum test of the COCO dataset.



(a) GT (b) 2 (c) 3

(d) 4 (e) 8 (f) 25

Figure 5.5: The ground truth vs. its reconstructions at various frequencies.

we incrementally added higher frequency coefficients and repeated the above procedure until we reached the maximum number of frequencies. Figure 5.4 shows the mean IoU loss at various frequencies. We can see that the loss decreases exponentially, and it is not saturating, and this is because this representation mathematically can represent any mask. Figure 5.5 illustrates a visual comparison between the ground truth and reconstructions using varying frequencies. Note how FourierMask reconstructions can deal with two masked instances.

5.2.4 Loss functions

The overall loss function comprises of three components, which is defined as:

$$L_{total} = L_{cls} + L_{box} + L_{mask} \tag{5.5}$$

We use the *focal loss* (Lin *et al.*, 2017b) for the classification loss L_{cls}, as given by equation 3.9. For the bounding box loss L_{box}, we use the smooth L1 loss. As for the mask loss L_{mask}, we tested two losses: the binary cross entropy loss, as defined in equation 3.10, and the IoU loss for training the binary masks defined as:

$$IoUloss = \frac{\sum_{i=0}^{N} min(y_{p_i}, y_{t_i})}{\sum_{j=0}^{N} max(y_{p_j}, y_{t_j})} \tag{5.6}$$

y_{p_i} is the predicted value of the pixel i, y_{t_i} is the ground truth value of the pixel i and N is the total number of pixels in the predicted mask. In the case of the MLP setup, we train a perceptron in parallel to the MLP. This parallel training helps, as the Fourier features we will pass to the MLP will be the Fourier series coefficients.

(a) Mask R-CNN (b) PointRend (c) FourierMask

Figure 5.6: Comparison between Mask R-CNN, PointRend and FourierMask.

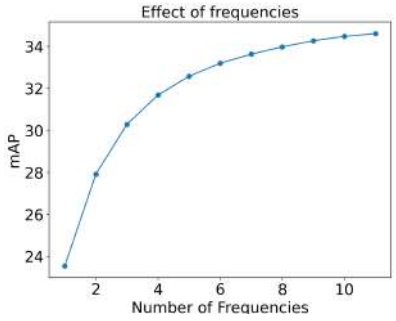
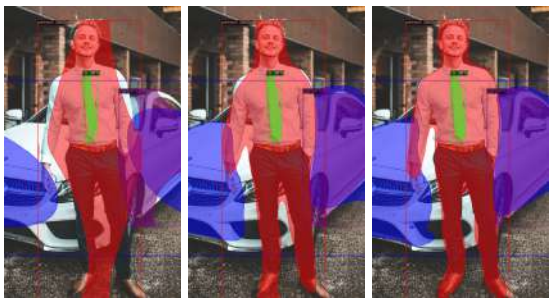


Figure 5.7: The mAP when using a subset of trained frequencies.



(a) 1 (b) 4 (c) 12

Figure 5.8: Mask predictions using various frequencies.

5.3 Experiments

For all our experiments we employ a Resnet 50 backbone with feature pyramid network pre-trained on ImageNet (Deng *et al.*, 2009) unless otherwise stated. We use the Mask R-CNN default settings from detectron2 (Wu *et al.*, 2019). We train on the MS COCO (Lin *et al.*, 2014) training set and show the results on its validation set. We predict class agnostic masks, i.e. rather than predicting a mask for each class in MS COCO, we predict only one mask per ROI. For the baseline, we trained a Mask R-CNN with class agnostic masks.

5.3.1 Number of frequencies

In this experiment, we want to see the effect of the number of frequencies on the performance. Therefore, we trained a FourierMask to predict a single vector of coefficients for the whole mask and we did not use the MLP. Figure 5.3a) shows the head architecture for this experiment. In the head, we applied two 3×3 strided convolutions to reduce the feature size by $1/4$ th and then used a fully connected layer to predict the coefficients. We

trained the network with twelve frequencies $N = 12$ and an output resolution of 56×56 using IoU loss. We evaluated the mAP precision of the network using a subset of Fourier frequencies, where we incrementally added the higher frequency components starting from the first component during inference. Figure 5.7 shows the result of this test. The mAP shows a similar trend as seen in figure 5.4 and therefore validates the spectrum analysis. Figure 5.8 shows how the mask changes when we use a different number of frequencies. We see that the mask gets more details as we increase the number of frequencies. Moreover, we trained the network using binary cross entropy loss rather than IoU loss. We used the same network architecture and settings. We reached a maximum of 32.1 mAP which was clearly lower than the 34.5 mAP using the IoU loss. Therefore, we used IoU loss for all the rest of our experiments.

5.3.2 Mask representation designs

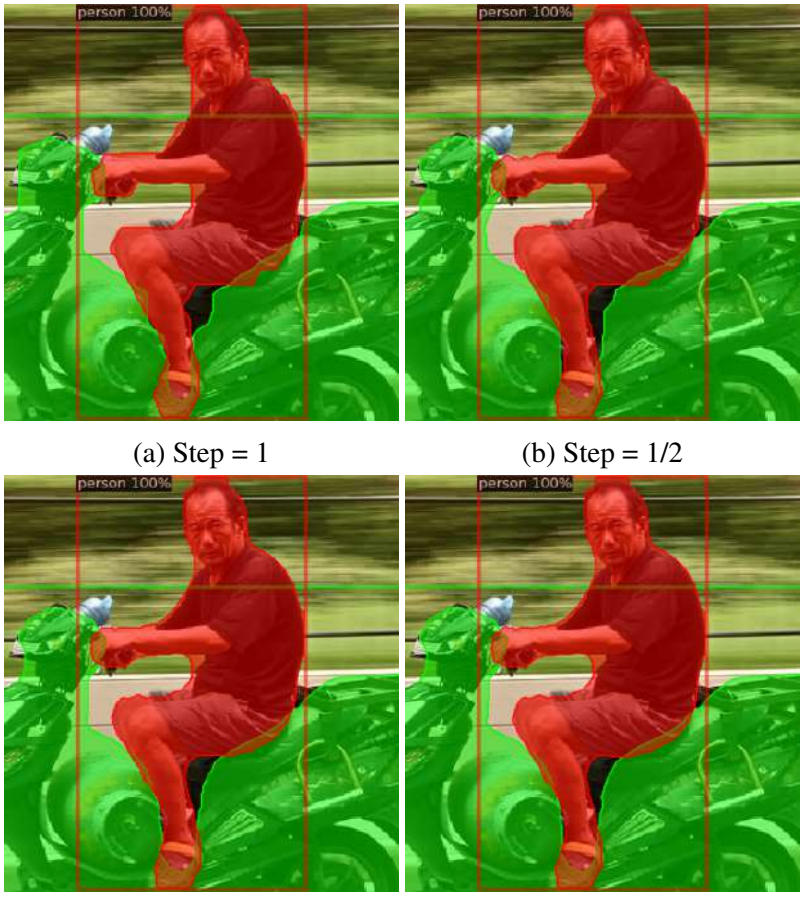
This section compares the different mask representation designs we proposed in section 5.2.2: A single Fourier series for the whole mask, multiple series one for each spatial location, and multiple MLPs one for each spatial location. in section 5.3.1, we trained a network that predicts a single Fourier series for the whole mask. In addition to that network, we trained two other networks with the architecture shown in figure 5.3b). In this architecture, the network predicts separate Fourier coefficients for each spatial location. In the first network we represented the mask patches as Fourier series, and the second network as a MLP (FM + MLP).

We used 12 Fourier frequencies for both networks and had an output resolution of 28 x 28 pixels. As for the MLP, we employed three hidden layers, where each layer had 256 neurons. The MLP has a single output neuron, on which we apply a sigmoid function to bound it between 0 and 1 to stabilize training. Furthermore, we investigated if sinusoidal activations in MLP perform better than a MLP with ReLU activations, as was claimed by Sitzmann *et al.* (2020).

Table 5.1: Comparison of various FourierMask's mask representation designs

Model	mAP
FM (single)	34.50
FM (multiple)	34.89
FM + MLP (ReLU)	34.41
FM + MLP (Sine)	**34.97**

Table 5.1 shows the results, and as can we see, predicting coefficients for each spatial location gives better performance, however, the difference is not big, compared to the added memory cost. In addition, networks with MLPs show the best performance among the models and sinusoidal activations showed better performance compared to



(a) Step = 1 (b) Step = 1/2

(c) Step = 1/4 (d) Step = 1/8

Figure 5.9: Subsampling the pixels smooths out the boundaries of the mask.

ReLU. Overall, we see the trend that whenever we increase the number of parameters to represent the mask we get better performance.

5.3.3 Higher resolution using pixel sub-sampling

One of the advantages of our mask representation is that it can predict masks at subpixel resolution, because implicit representations are continuous in the input domain. We analyzed this by evaluating both the trained networks in section 5.3.2 on the MS COCO validation set on various pixel steps and compared it to our baseline MaskRCNN. For the input x in the equation 5.2, rather than using integer values of pixels (pixel step of 1), we used a pixel step of $1/2^{s-1}$, where $s \in \mathbb{Z}^+$ is the scaling factor. This effectively scaled both the height and width of the input x by a factor of s. Table 5.2 shows the results, we can see that FourierMask surpasses our baseline, which means that our representation is better than the grid representation. Furthermore, we can observe that sub-sampling

Table 5.2: The effect of sub-sampling the pixels during inference. The speed is tested on Nvidia GTX 2080Ti GPU

Model	Pixel Step	Resolution	mAP	Speed (ms)
Mask R-CNN	1	28×28	34.86	48.7
FM	1	28×28	34.89	50.3
FM	1/2	56×56	35.13	59.1
FM	1/4	112×112	**35.18**	68.3
FM + MLP	1	28×28	34.97	52.1
FM + MLP	1/2	56×56	**35.18**	67.0
FM + MLP (ResNeXt-101)	1	28×28	39.09	-

always improves the mAP, and widens the gap with the grid representation even more. Figure 5.9 shows how the mask boundary smooths out when sub-sampling the pixels. Note that we trained the network on a 28×28 output resolution, but we can generate higher resolution output during inference, which is a considerable advantage over other methods.

5.4 Conclusion

In this chapter, we showed how implicit representations combined with the Fourier series can be applied to the task of instance segmentation to generate high-quality masks. We illustrated that the masks generated using our Fourier mapping are compact and meaningful. The lower Fourier frequencies hold the shape and higher frequencies hold the sharp edges. Furthermore, by sub-sampling the pixel coordinates in our implicit MLP, we can generate higher resolution masks during inference, which are visually smoother and improve the mAP over our baseline Mask R-CNN with similar settings and model capacity.

Chapter 6

3D multi-object tracking

6.1 Introduction

In the first part of the thesis, we focused on instance segmentation. Now we will focus on the second task, which is 3D multi-object tracking (MOT). 3D MOT aims to find the objects surrounding an agent in 3D space and trace them through time. The trajectories built by the MOT algorithms are used by motion forecasting modules or given directly to the motion planner to complete navigation successfully. This chapter introduces the 3D multi-object tracking task and describes the approaches to solving the task. In addition, we explain the pipeline of online MOT and introduce the most popular 3D multi-object tracking datasets and the evaluation metrics of this task. It gives the prerequisites for chapter 7, in which we describe our own methods.

6.2 Approaches to 3D multi object tracking

In general, we divide MOT in terms of data association into batch tracking and online tracking; batch MOT aims to find the global data association by finding the minimum cost of a flow graph (Schulter *et al.*, 2017). In online MOT, sometimes called tracking-by-detection, the goal is to perform data association of the last two frames only, where it becomes a bipartite matching problem. It is usually solved using the Hungarian algorithm (Kuhn, 1955). For 3D MOT, online MOT has become popular nowadays because of its simplicity, efficiency, and influence of the 2D MOT methods (Bergmann *et al.*, 2019; Bewley *et al.*, 2016; Wojke *et al.*, 2017; Zhou *et al.*, 2020a). These algorithms filter outliers in frame-by-frame object detectors (Lang *et al.*, 2019; Shi *et al.*, 2019b; Cheng *et al.*, 2020) by utilizing temporal information. Also, because they rely heavily on object detection performance, the notable advancement in 3D object detection resulted in considerable MOT growth.

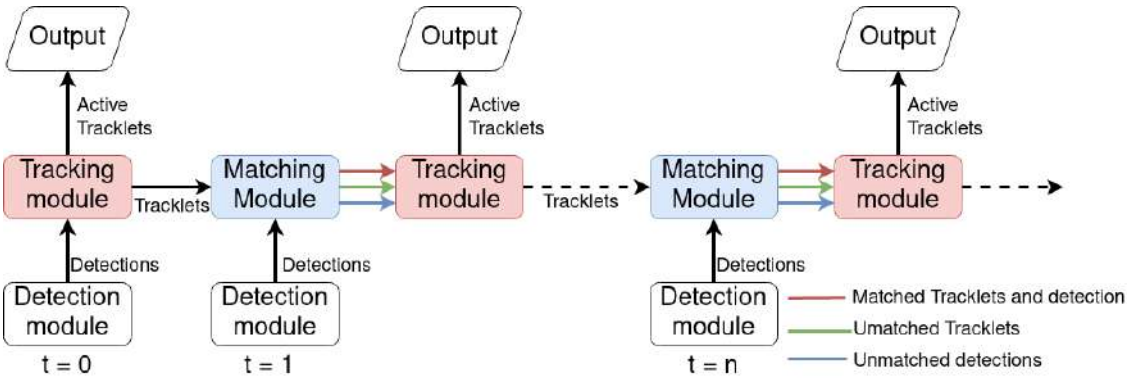


Figure 6.1: he general pipeline for an online MOT system. It consists of three main modules: 1) The detection module, 2) The matching module, 3) The tracking module.

6.3 multi object tracking pipeline

The general pipeline for 3D MOT has the structure shown in figure 6.1. It has a detection module, a matching module, and a tracking module. In the following sections, we will describe the modules in the pipeline.

6.3.1 Detection module

The detection module consists of a 3D object detector. It processes point clouds and/or images and returns 3D bounding boxes. At each time step t, the detector produces a number n of detections $D^t = \{D_1^t, D_2^t, \ldots, D_n^t\}$. Each detection D_i^t is a tuple of 8 parameters $(x, y, z, l, w, h, \theta, s)$. Here, the parameters (x, y, z) are the object's center in either the vehicle's local frame or the global frame, the parameters (l, w, h) are the bounding box dimensions, θ is the yaw angle, and s is the detection score. A 3D MOT system's performance is mainly affected by object detection performance (Weng and Kitani, 2019; Weng *et al.*, 2020).

One of the first algorithms trained end-to-end on point clouds for the 3D detection task was VoxelNet (Zhou and Tuzel, 2018). It used PointNets (Qi *et al.*, 2017) to produce point features inside each voxel. They process these features with 3D sparse convolutions, then passed into a region proposal network. Finally, they use 2D convolutions to produce the detections. SECOND (Yan *et al.*, 2018) sped up VoxelNet by using spatially sparse convolutional networks to extract features from the z-axis before they downsample the 3D data to 2D features. More techniques attempted to eliminate the costly 3D convolutions. For example, PIXOR (Yang *et al.*, 2018) projected all points onto a 2D feature map with 3D occupancy and point intensity data encoded in the feature dimension. PointPillars (Lang *et al.*, 2019) used pillars instead of voxels to give pillar features; it used 2D convolutions on the pillar feature, which provided an efficient backbone.

PointRCNN (Shi *et al.*, 2019a) took another approach and avoided voxelization and directly operated on 3D point clouds. They generate proposals and then refine them to generate the detections using PointNets. PV-RCNN (Shi *et al.*, 2020) merged the voxel-based and point-based pipelines to get the advantages of both. Other works focused more on discovering helpful features for detection. For example, MVF (Zhou *et al.*, 2020b) fused features from bird's-eye view (BEV) and perspective views of the same lidar point cloud and introduced the concept of dynamic voxelization, where there is no need to set a fixed amount of points per voxel. PointPainting (Vora *et al.*, 2020) added semantic segmentation information to point clouds as visual features. Hu *et al.* (2020) used free space as additional information to the detector. CenterPoint (Yin *et al.*, 2020) considered object detection as a keypoint estimation problem as was done in (Zhou *et al.*, 2019c) for 2D detection. It predicts a heatmap for each class and then uses the heatmap's local minima features to regress to 3D bounding boxes.

6.3.2 Matching module

This module is responsible for matching the detected 3D bounding boxes and the tracking module estimations. There are mainly two algorithms for this task; the Hungarian algorithm or a greedy algorithm (Chiu *et al.*, 2020; Yin *et al.*, 2020). Both need a metric to perform the matching, and examples of the metric used are IoU (Weng and Kitani, 2019), euclidean distance (Yin *et al.*, 2020), Mahalanobis distance (Chiu *et al.*, 2020), and distance in embedding space (Weng *et al.*, 2020; Baser *et al.*, 2019). If the metric is above a threshold, we consider the detection and the tracklet matched; otherwise, they are unmatched.

AB3DMOT (Weng and Kitani, 2019) built a simple pipeline where they used a Kalman filter (Kalman *et al.*, 1960) for tracking objects and the Hungarian algorithm for data association between the tracklets and detections. They used IoU as a matching metric and showed a great performance in terms of speed and accuracy. Chiu *et al.* (2020) used the Mahalanobis distance (Mahalanobis, 1936) instead of the IoU. They also found the Kalman filter parameters automatically from the statistics of the detector's training results. FANTrack (Baser *et al.*, 2019) integrated 2D appearance features from CNNs and 3D bounding box features for data association. GNN3DMOT (Weng *et al.*, 2020) proposed using four features: motion and appearance from 2D and 3D spaces, and used graph neural networks to learn the interaction between these features.

6.3.3 Tracking module

The tracking module is responsible for four tasks: to give new tracklets an ID and save them in memory, decide whether a tracklet should be *active* or not, estimate the tracklet's parameters in the next time step, and update the tracklet's information according to its matched detection using a filtering algorithm (e.g. a Kalman filter).

The tracklets' state depends on the filtering algorithm used. However, it should have at least ten parameters $(x, y, z, l, w, h, \theta, id, c, active)$. The first seven parameters are the first seven parameters of the detection tuple discussed in section 6.3.1. *id* is an identifier that is unique for every instance in the sequence. c is the tracklet score which tells how confident the system is about the tracklet. And *active* is a Boolean variable stating whether the tracklet is active or not. When a tracklet is not *active*, the tracking module does not consider it an output, yet keeps it in memory. The module sets the tracklet's *active* variable to true if it is matched with a detection or it is newly created and has a score higher than the *detection threshold*. Also, it sets the *active* variable of unmatched tracklets kept by the object death module to false if their score is lower than the *active threshold*. *Min-hits* generally lowers false positives and increases false negatives, and *max-age* enhances robustness against missed detections and occlusions but increases false positives.

All unmatched tracklets from the matching module are sent to the death module to decide whether to keep or delete the tracklet. The decision can be *count-based*, *confidence-based* or a mix of both. In *count-based*, the module deletes the tracklet if it was unmatched for a number (*max-age*) of timesteps. In *confidence-based*, the module deletes the tracklet if its score goes below the *deletion threshold*. We can use a combination of both; however, one will usually dominate the performance.

(a) KITTI (b) NuScenes (c) Waymo

Figure 6.2: The cars used to collect the autonomous driving datasets. All of the cars contain cameras and at least one LIDAR.

6.4 Datasets

6.4.1 KITTI

The KITTI vision benchmark suite (Geiger *et al.*, 2012) was one of the first large scale datasets for autonomous driving perception tasks such as stereo, optical flow, visual

odometry, SLAM, 3D object detection, and 3D multi object tracking, which we are intrested in. They recorded the data using a Volkswagen Passat that is shown in figure 6.2a). It is equipped with two color cameras, two grayscale cameras (resolution: 1392 × 512), a Velodyne HDL-64E LIDAR, and a OXTS RT 3003 localization system which combines GPS, GLONASS, an IMU and Real-time kinematic (RTK) positioning. They recorded the data during sunny weather at the city of Karlsruhe, Germany. The dataset has 22 scenes that cover 39.2km, and has more than 200k 3D abject annotations with 28 different classes. Figure 6.3a) shows some examples from KITTI, illustrating the different situations it captures. Until recently, KITTI was still used to benchmark some tasks but it is currently out-competed by other larger datasets like nuScenes and Waymo's open dataset.

6.4.2 NuScenes

The nuScenes dataset (Caesar *et al.*, 2020) is one of the largest datasets used for benchmarking autonomous driving tasks. The dataset is developed by Motional, and it benchmarks 3D detection, 3D multi object tracking, motion prediction, LIDAR segmentation, and planning. They used two Renault Zoe electric cars as shown in figure 6.2b), with an identical sensor layout to drive in Boston and Singapore. The cars are equiped with six cameras covering the 360° field-of-view (resolution: 1600 x 900), a 32 beam LIDAR, five 77GHz Frequency-modulated continuous-wave (FMCW) RADARs, IMU, and GPS with RTK positioning. They recorded 1000 driving scenes, each with a duration of 20 seconds captured in Boston and Singapore, and have 23 object classes annotated with 3D bounding boxes at a rate of 2Hz. They recorded at different times of the day and figure 6.3b) shows some examples.

6.4.3 Waymo's open dataset

Waymo's open dataset is the largest dataset used for benchmarking autonomous driving tasks, and it is developed by Waymo. It benchmarks 3D detection, 3D multi object tracking, motion prediction, LIDAR segmentation, occupancy and flow prediction. They used the car shown in figure 6.2c) which is equiped with five cameras covering about 270° field-of-view (main camera resolution: 1920 x 1280), five LIDARs, and an IMU. They recorded 1150 driving scenes, each with a duration of 20 seconds captured in San Fransisco, Mountian View, and Phoenix. The dataset has four object classes (Vehicle, Pedestrian, Cyclist, and Sign) annotated with 3D bounding boxes at a rate of 10Hz. They recorded at different times of the day and figure 6.3c) shows some examples. Compared to nuScenes, Waymo has more annotations because of the higher rate, covers more geographical area, and has higher resolution LIDARS. On ther other hand nuScenes's cameras cover 360°, it has RADAR data, and has a more challenging multi object tracking challenge because of the lower rates. There are more datasets that we can use for 3D multi object tracking which are smaller in size than NuScenes and Waymo or with less

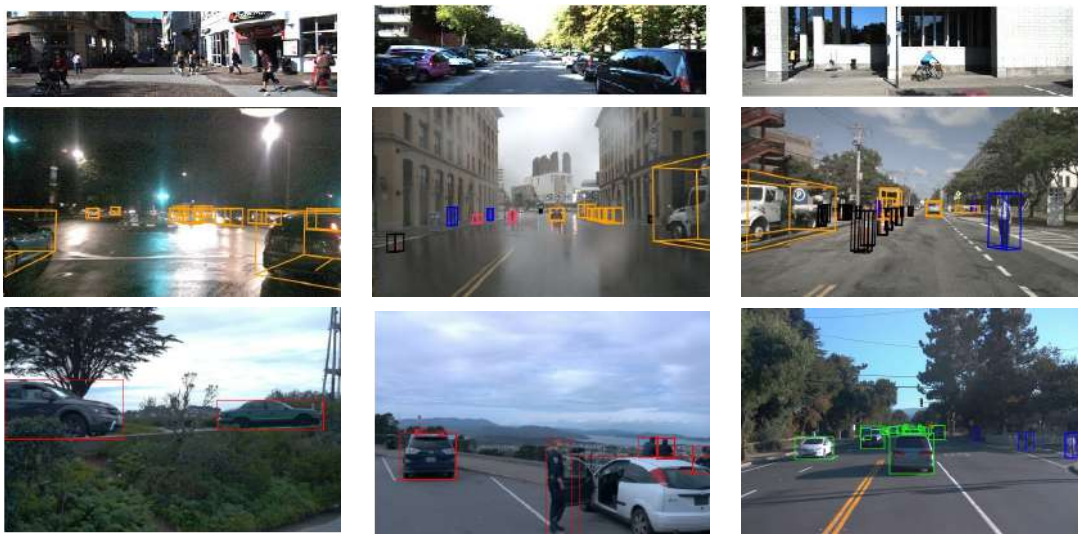


Figure 6.3: Samples of images from the autonomous driving datasets (rows from top to bottom): a) KITTI, b) NuScenes, c) Waymo's open dataset.

labels, but are worth mentioning for example: Argoverse (Chang *et al.*, 2019), ONCE (Mao *et al.*, 2021), Lyft L5 datasets (Houston *et al.*, 2021). each of which having it own advantages and disadvantages.

6.5 Evaluation

The primary metrics used for evaluating multi object tracking are the Multi-Object Tracking Accuracy (*MOTA*), Average Multi-Object Tracking Accuracy (*AMOTA*), and Average Multi-Object Tracking Precision (*AMOTP*). *MOTA* is expressed as:

$$MOTA = 1 - \frac{FP + FN + IDS}{GT} \tag{6.1}$$

FP is the number of false positives, FN is the number of false negatives, IDS is the number of identity switches, and GT is the number of ground truths. *MOTA* is found after fine-tuning the thresholds for each class, and then the maximum value is reported. On the other hand, *AMOTA* measures the average performance of different thresholds, and it is expressed as:

$$AMOTA = \frac{1}{n-1} \sum_{r \in \{\frac{1}{n-1}, \frac{2}{n-1} \dots 1\}} MOTAR \tag{6.2}$$

where r is the recall value, n is the number of recall values to test on, and *MOTAR* is the recall-normalized *MOTA*, and it is given as:

$$MOTAR = \max\left(0, 1 - \frac{IDS_r + FP_r + FN_r + (1-r)\cdot GT}{r \cdot GT}\right) \tag{6.3}$$

Here IDS_r, FP_r, and FN_r are the id switches, false positives, and false negatives at a specific recall value r. The *AMOTA* is a metric that favors algorithms that are robust against changes in the evaluation thresholds. As for the AMOTP, we define it as:

$$AMOTP = \frac{1}{n-1} \sum_{r \in \{\frac{1}{n-1}, \frac{2}{n-1}, .., 1\}} \frac{\sum_{i,t} d_{i,t}}{\sum_t TP_t} \tag{6.4}$$

Here $d_{i,t}$ indicates the position error of track i at time t, and TP_t indicates the true-positives at time t.

Chapter 7

Score refinement for confidence based 3D multi object tracking

7.1 Introduction

In online multi object tracking (MOT), the decision of when to initialize and terminate tracklets is critical to its performance. In the previous works (Weng and Kitani, 2019; Chiu *et al.*, 2020), this decision was *count-based*. Those methods only make the tracklet active if it has a minimum number of consecutive detection matches (*min-hits*). On the other hand, they terminate a tracklet if it is not matched for a predefined number of timesteps (*max-age*). The problem with the count-based method is that it treats all detections the same, yet the detection score can indicate the detection quality. For example, if the detection score is 0.95, it is probably a true positive. In this case, why should the algorithm wait for more matches if it is already confident about the first detection? Moreover, if the tracklet's initial score is low, why should it remain for many timesteps before we terminate it?

Furthermore, when a detection matches a tracklet, the tracklet's updated score is the detection score. Here the tracklet's previous time step score is not used, and we believe it is a valuable information source that can improve the score estimation. For these reasons, we use a *confidence-based* method for initialization and termination.

The *confidence-based* method initializes a tracklet and considers its output when its score is higher than the detection threshold (*det-th*). It terminates it when its score goes below the deletion threshold (*dlt-th*). Moreover, the tracklet's score decreases in the estimation step by a constant value (*score-decay*), and if it is matched with a detection, it increases by the *score update function*. In this case, tracklets consistently matched over time will have high scores, and unmatched tracklets' scores will decline. Sun *et al.* (2019) was the only work that used *confidence-based tracking*. They added the detection and tracklet scores to update the tracklet's scores.

Our work will show that their *score update function* performance is poor, and in the best cases, it will work like the *count-based* method. Consequently, we will show that *confidence-based* MOT outperforms *count-based* MOT if we employ proper *score update functions*. We propose score update functions by our intuition, and at the same time,

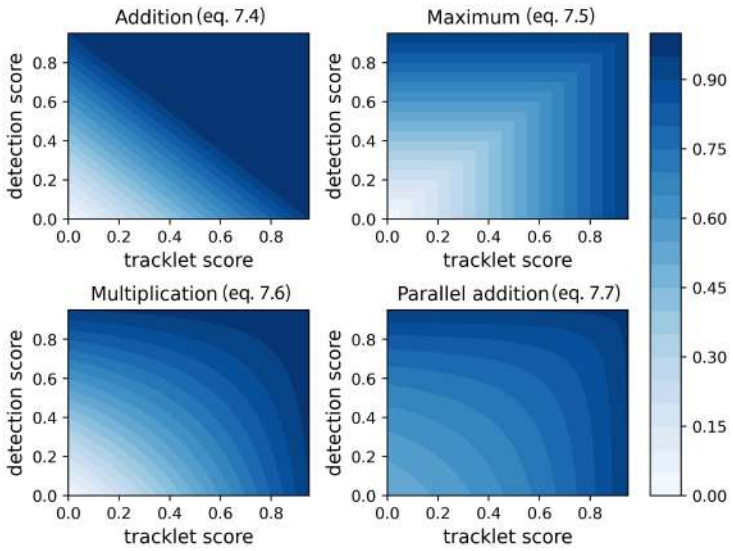


Figure 7.1: Contour plots of the different update function outputs.

we propose learning them using MLPs. In addition, we found that our method worked well in refining scores in the time domain, so we decided to refine scores across different modalities and used it as a late fusion ensemble method. This chapter is based on two works: our paper "Score refinement for confidence based 3D multi-object tracking" (Benbarka *et al.*, 2021) and the bachelor's thesis "Learning score update functions for confidence-based multi-object tracking" by (Gherri, 2021) supervised by the author.

7.2 Method

This section will explain how we refine the tracklet's scores. In *count-based* methods, the tracklet score is simply the score of the matched detection, as shown in equation 7.1

$$c_t = s_t. \tag{7.1}$$

c_t is the tracklet score, and s_t is the matched detection's score of the current time step t. And if there is no matched detection, the tracklet score is unchanged $c_t = c_{t-1}$. We think that this approach has two disadvantages. First, we believe that the algorithm should believe less in the tracklet presence by time unless continuously matched with detections. For this reason, the score update module reduces the tracklets' score by a constant *score-decay* σ_{score} to get an estimated score of the current time step, as shown in equation 7.2.

$$\hat{c}_t = c_{t-1} - \sigma_{score} \tag{7.2}$$

here $\hat{c}_t$ is the estimated tracklet score of the current time step. The second disadvantage of the *count-based* method is that the new tracklet score does not depend on the previous time step's score c_{t-1} if there is a match. We believe that c_{t-1} is a valuable information source that can better measure the current score c_t. Therefore, we use update functions that are dependant on the previous tracklet and detection scores. Furthermore, we argue that if there is a match, the algorithm should be more confident about its decision than its detection's confidence or the previous time step's confidence. Accordingly, we set the criterion for selecting an update function to give a score greater than or equal to both the tracklet and detection scores, as shown in equation 7.3.

$$c_t = f(\hat{c}_t, s_t) \geq max(\hat{c}_t, s_t) \tag{7.3}$$

One function that satisfies this criterion is adding the tracklet and detection scores (equation 7.4), as was used by Sun *et al.* (2019).

$$c_t = \hat{c}_t + s_t \tag{7.4}$$

We reason that adding the scores will make the algorithm overconfident, and as seen in figure 7.1, it is quite saturated. To tackle this problem of overconfidence, we propose the functions given in equations 7.5, 7.6, and 7.7, which satisfy the above criterion and provide more robust scores.

$$c_t = max(\hat{c}_t, s_t) \tag{7.5}$$

$$c_t = 1 - ((1 - \hat{c}_t) \cdot (1 - s_t)) \tag{7.6}$$

$$c_t = 1 - \frac{(1 - \hat{c}_t) \cdot (1 - s_t)}{(1 - \hat{c}_t) + (1 - s_t)} \tag{7.7}$$

For equation 7.5, it is the minimum requirement to satisfy the criterion above. And as for equations 7.6 and 7.7, they are seeking to decrease the scores' complements. The intuition is that if the score indicates the detection's confidence, then the score's complement indicates its uncertainty. If we multiply (eq. 7.6) or parallel add (eq. 7.7) these uncertainties, they become smaller, so naturally, the confidence gets higher. These equations give a middle ground between adding the scores and taking the scores' maximum, as shown in figure 7.1. Finally, the new score of unmatched tracklets is the estimated score $c_t = \hat{c}_t$.

Finally, we need to decide whether to keep or delete unmatched tracklets from the matching module. In *count-based*, the module deletes the tracklet if it was unmatched for a number (*max-age*) of timesteps. In *confidence-based*, the module deletes the tracklet if its score goes below the *deletion threshold*. We can use both; however, only one will dominate the performance.

7.2.1 MLP as a score update function

The score update functions we used in section 7.2 are not learned but rather chosen by intuition. We believe that using an MLP as an update function could outperform these functions because of its learning capabilities. The MLP task is to update the matched tracklets score using at least two scores, the estimated tracklet score and the detection score of the matched detection in the current time step. The multi-layer perceptron input has at least two variables: two scores, $\hat{c}_t$ and s_t, and it has one output, a newly updated tracklet score c_t. The MLP can have more inputs, such as object classes.

We need to train the MLP to accomplish this task. Therefore, we need to generate training data. For gathering the train data, we use the following pipeline as shown in figure 7.2.

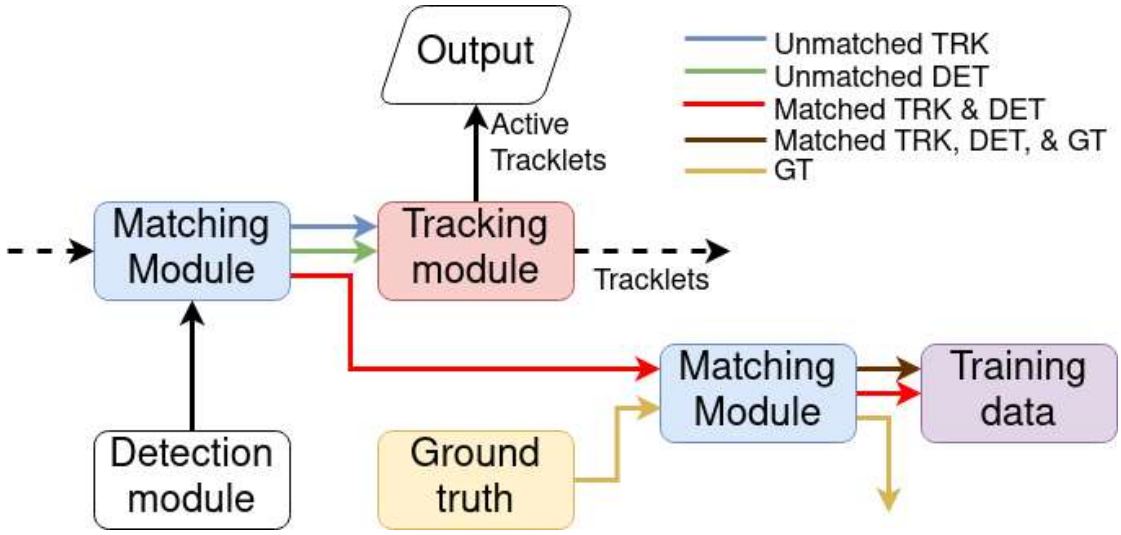


Figure 7.2: An overview diagram of the pipline used to generate training data for the MLP score update function.

First, we run the tracking system, then take the matched detections and tracklets and pass them to another matching module to match them with the ground truth labels of the dataset. We only take the matched detections and tracklets because we only apply the score update function to them in a running system. As a result, the outputs of the second matching module are divided into three cases:

1. Matched detections, tracklets, and ground truth.
2. Matched detections and tracklets but unmatched with ground truth.
3. Unmatched ground truth.

We take the scores of the first case and set their labels to 1, and we take the scores of the second case and set their labels to 0. We do not use the third case to generate data

because it does not have detections and tracklets. After training the MLP on this data, it should manage to update the score appropriately.

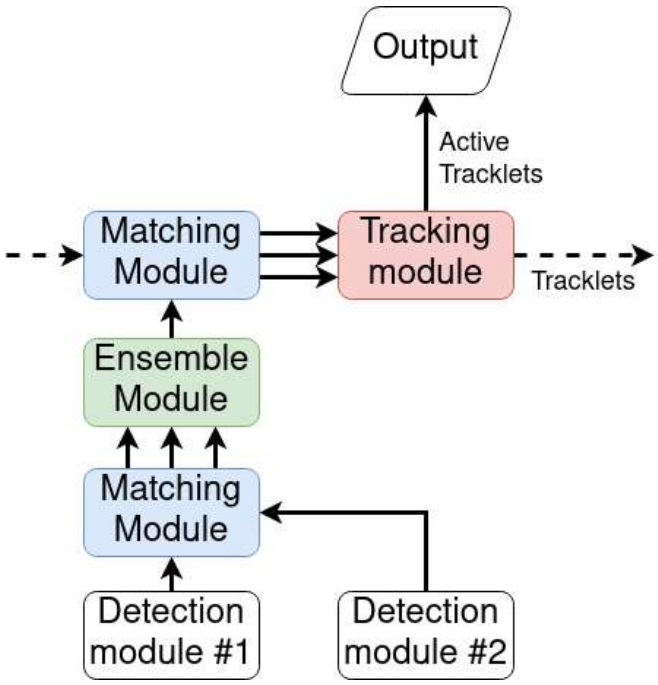


Figure 7.3: An overview diagram of the pipline used in late fusion of detection modules.

7.2.2 Late-fusion ensemble method

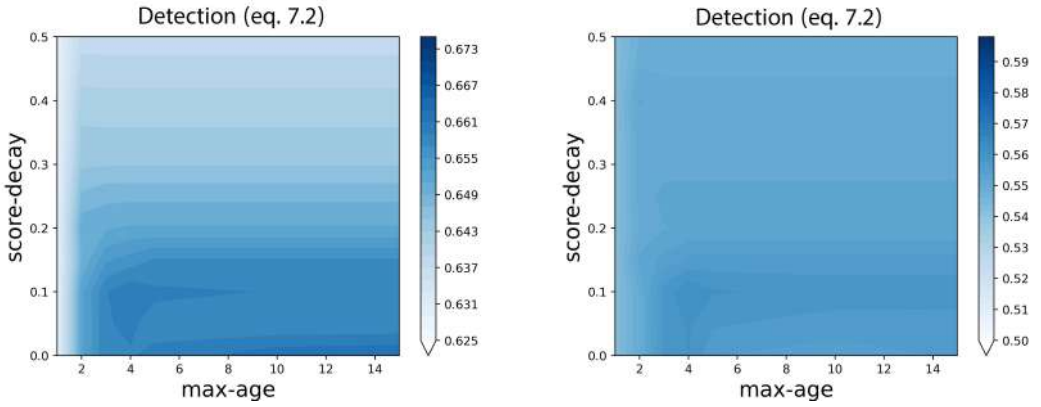
Another way to apply score refinement is to use it as a late-fusion ensemble method. We apply it to detections of two different modalities rather than tracklets and detections of two consecutive time steps, as shown in figure 7.3. Whenever two modalities detect the same object, the confidence of its tracklet rises; otherwise, it drops. For comparison, other ensemble methods use voting strategies (Casado-Garcıa and Heras, 2020): the affirmative strategy takes all proposals, the consensus strategy takes those recognized by the majority, and the unanimous strategy takes those identified by all modalities. Since we only have two modalities, the consensus and unanimous strategies are the same here.

7.3 Experiments

7.3.1 Ablation study

We did our tests on the nuScenes dataset (Caesar *et al.*, 2020) and used CenterPoint (Yin *et al.*, 2020) as a 3D detector and its point tracker as the filtering algorithm. We

choose the greedy algorithm as the matching algorithm and the euclidian distance as the matching metric. We began with the following hyperparameters: *max-age* = 3, *score-decay* = 0.0, *deletion threshold* = 0.0 , *active threshold* = 1.0, *detection threshold* = 0.0, *min-hits* = 1 and the *update function* was equation 7.1. This configuration produced the best results as reported in CenterPoint, and we made it our starting point for the ablation study. The next sections will discuss the impact of each hyperparameter.



(a) AMOTA (b) MOTA

Figure 7.4: Contour plots of a) the AMOTA and b) the MOTA results for the *score-decay* experiment.

Score-decay and max-age

We performed a grid search with the values of (0, 0.05, 0.1, 0.15, 0.2, 0.25, 0.3, 0,4, 0.5) for *score-decay* and (1, 2, 3, 4, 5, 15) for *max-age*, and figures 7.4a and 7.4b show a contour of the AMOTA and MOTA results, respectively. We observe that the system gives high AMOTA and MOTA results at low *score-decay* and large *max-age*. We also see when we increase the *score-decay* or lower the *max-age*, the performance deteriorates.

The reason is that when we use equation 7.1 as a score updating function, the detection score overwrites the tracklet score. Thus, changing the *score-decay* essentially affects the tracklet's lifetime, which is similar to changing *max-age*. However, there is a slight difference between *score-decay* and *max-age* impacts; tracklets with low scores are deleted faster with *score-decay* than with *max-age*. This difference gave a 0.2 MOTA improvement at a *score-decay* of 0.1; however, this improvement is not significant.

Score update

We repeated the experiment in section 7.3.1 but with the other score-update functions (equations 7.4, 7.5, 7.6, 7.7). Figures 7.5 and 7.6 show the AMOTA and MOTA results,

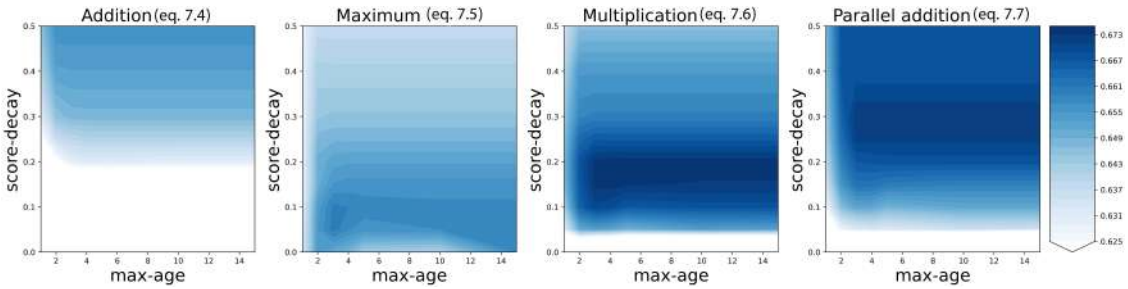


Figure 7.5: Contour plots of the AMOTA results of the different score update functions.

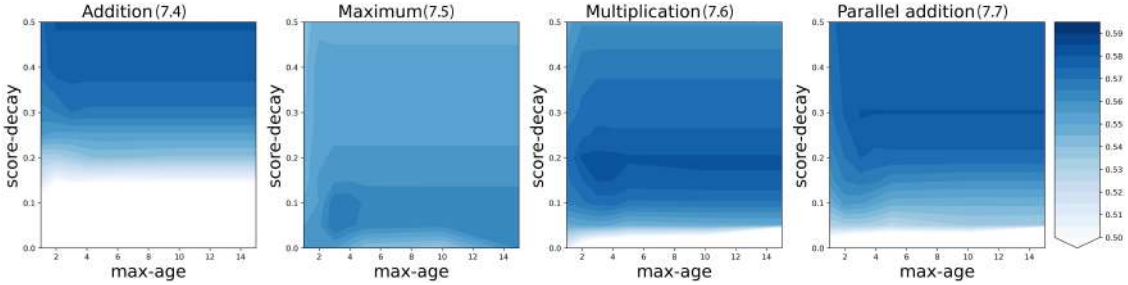


Figure 7.6: Contour plots of the MOTA results of the different score update functions.

respectively. We notice that for equations 7.5, 7.6, and 7.7, when we increase the *max-age*, the performance improves until it peaks at a value of 4 and remains constant. As for equation 7.4, the performance is dependent on *max-age* even at high values. However, we also see that it performs worse than the other functions, and we can infer that it is not a proper function for the task. Moreover, we can conclude from these results that *max-age* can be neglected.

We also observe that when the *score-decay* value is approaching zero, the performance is terrible, and this is more predominant in equations 7.4 and 7.7, which increase the score a lot in one update. From this observation, we can conclude that using update functions without the *score-decay* does not work, and therefore, they should be applied together.

Furthermore, we perceive that the contour's shape is an inverted parabola in the *score-decay* direction for all update functions. However, the difference between the functions is the value and position of the inverted parabola's peak. The more the update function can increase the score in one update step, the higher the *score-decay* it needs to peak. And in the extreme case of equation 7.4, it needs a *score-decay* of 0.5 to peak. This means that it will most likely floor the estimated tracklet score to zero before adding it to the detection score. In this case, the algorithm works similarly to *count-based* methods, where the detection score overwrites the estimated tracklets score, making it a poor update function to choose.

To further investigate what happens when changing the *score-decay*, we plotted the metric details of class 'CAR' when using equation 7.6 as an update function in figure

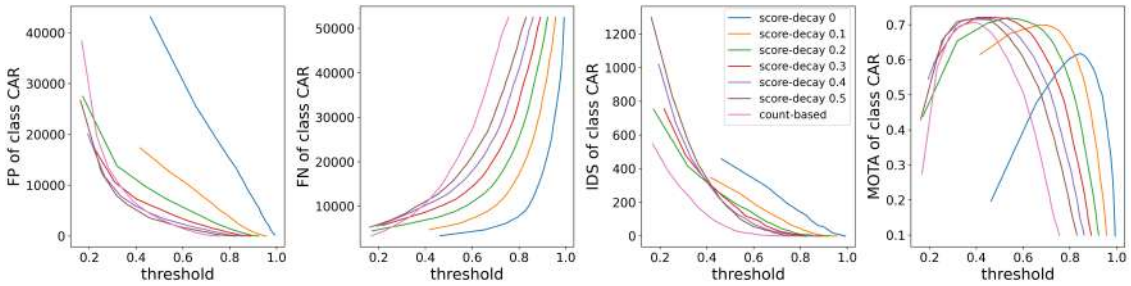


Figure 7.7: Metric details of class CAR vs score for different score-decay values.

7.7. We use the initial hyperparameters mentioned in section 7.3.1 for the baseline. We observe from figure 7.7 that when we increase the *score-decay*, the whole FN curve shifts to the left, which results in shifting the MOTA peak to higher thresholds and lowering its value. On the other hand, increasing the *score-decay* reduces the FP and IDS curves' slope, which results in increasing the MOTA peak value and shifting its position to lower thresholds. The MOTA curve shows that its peak moves to lower thresholds and increases its peak value. This means that the *score-decay* affects FP and IDS more than FN, which explains why our method works. In the rest of the ablation study, we pick equation 7.6 as the score update function, set max-age to infinity (or not use it), and use a simple line search to find *score-decay*.

The rest of the hyperparameters

We tested the rest of the hyperparameters, and we found that increasing *min-hits* or the *deletion-threshold* always reduces the performance. As for the *active-threshold*, we found that decreasing it to 0.75 improved AMOTA by 0.03. And for the *detection-threshold*, we found that increasing it to 0.15 further improved AMOTA by 0.05. We can conclude from the ablation study that our method has only one main hyperparameter, which is the *score-decay*, and we can find its optimal value easily with a simple line search. The rest of the hyperparameters are either not needed (*max-age*, *min-hits*, and *deletion-threshold*) or do not dramatically affect the performance (*active-threshold* and *detection-threshold*).

7.3.2 Method generalization

After finishing the ablation study, we wanted to check if our method generalizes well. We took the best configuration from the ablation study and applied it to different filtering algorithms and detectors. We used CBGS (Zhu *et al.*, 2019) as the second detector and Kalman Filtering as a second filtering algorithm. The state of the Kalman filter is a 6D vector consisting of the center position, velocity, and acceleration in the x and y directions. Tables 7.1 and 7.2 show that score refinement always improves the results

Detector	Tracker	SR	AMOTA↑	MOTA↑	FP↓	FN↓	IDS↓
CenterPoint	PointTracker	-	65.88	56.01	**12295**	21546	**479**
CenterPoint	PointTracker	✓	**67.51**	**58.3**	13666	**18882**	494
CenterPoint	KalmanFilter	-	65.39	55.33	13307	20559	811
CenterPoint	KalmanFilter	✓	**67.22**	**58.29**	**13072**	**19534**	**610**
CBGS	PointTracker	-	60.13	51.82	**10729**	24116	**754**
CBGS	PointTracker	✓	**61.66**	**54.20**	11408	**22518**	764
CenterPoint*	PointTracker	-	63.84	53.66	18612	**22928**	760
CenterPoint*	PointTracker	✓	**64.93**	**54.51**	**16469**	24092	**557**

Table 7.1: Summary of the tracking results on the nuScenes dataset. SR is short for Score Refinement. *evaluated on the test split

Detector	Tracker	SR	MOTA↑	FP↓	FN↓	IDS↓
CenterPoint	PointTracker	-	48.21	9.57	**41.06**	1.16
CenterPoint	PointTracker	✓	**48.53**	**9.41**	41.31	**0.75**
CenterPoint	KalmanFilter	-	50.15	**9.75**	39.91	0.19
CenterPoint	KalmanFilter	✓	**50.76**	9.87	**39.25**	**0.12**
PointPilllars	KalmanFilter	-	40.51	**9.69**	49.66	0.14
PointPilllars	KalmanFilter	✓	**41.01**	10.19	**48.7**	**0.1**

Table 7.2: Summary of the tracking results on the Waymo dataset. SR is short for Score Refinement.

even without further tuning the hyperparameters. We also uploaded our tracking results on the test split on the evaluation server, and we got an improvement of 1.1 in AMOTA against CenterPoint (Yin *et al.*, 2020). We say these improvements are consistent and computationally free, so there is no reason not to use score refinement.

7.3.3 Max-distance threshold

The matching module uses the max-distance threshold to determine if two detection and tracklet pairs are matched. We took CenterNet (Yin *et al.*, 2020) with the count-based method as a baseline for the experiment. After finishing the previous work in this chapter, we noticed a possibility for improvement by changing the max-distance threshold of each class, so we did it, and we got a considerable AMOTA improvement. Table 7.3 illustrates this improvement.

max-distance threshold change					
State	AMOTA↑	MOTA ↑	FP↓	FN↓	IDS ↓
Before	65.9	56	13317	20274	562
After	**67.1**	**57.5**	**12771**	**20254**	**544**

Table 7.3: Result of changing the max-distance threshold. evaluated on the val split

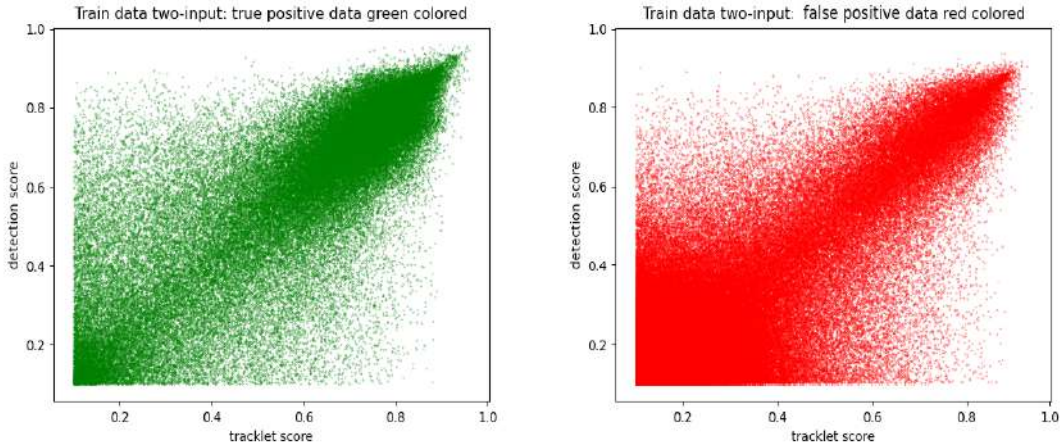


Figure 7.8: The training data for the MLP score update function. The figure on the left with the green points are the data points that are labeled as one, and the red points on the right are labeled as zero. We can see clearly that the classes are not separable.

7.3.4 MLP as score update function experiments

To generate the MLP's training data, as discussed in section 7.2.1, we ran CenterPoint with the hyperparameters that performed best for the count-based method. We used the point tracker as the filtering algorithm. We choose the greedy algorithm as the matching algorithm and the euclidean distance as the matching metric.

As for the rest of the hyperparameters: *max-age* = 3, *score-decay* = 0.0, *deletion threshold* = 0.0 , *active threshold* = 1.0,*detection threshold* = 0.0, *min-hits* = 1 and the update function was equation 7.1.

Figure 7.8 shows the resulted training data. The green points are data points that have a label of 1 (Case 1: matched detections, tracklets, and ground truth), and the red points have a label of 0. We can see three observations from this figure. First, there is a significant overlap between the two classes, and they are not separable. Second, zero-labeled points are three times bigger (≈ 300000) than those one-labeled points (≈ 90000).

The final notice is that both sets have an maximum which is less than one, the red points have a maximum of ≈ 0.94, and the green points have a maximum of ≈ 0.96.

Method	Activation	AMOTA	MOTA	FP	FN	IDS
Count-based	-	67.1	57.5	**12771**	20254	544
MLP score update function	Tanh	68.8	58.9	12850	20534	**423**
	Sine	68.4	58.8	13476	19592	495
	Relu	69.3	**59.9**	13152	19332	481
	LeakyRelu	**69.4**	59.6	13543	**18896**	505

Table 7.4: Summary of the tracking results on the val split of the nuScenes dataset using MLP as a score update function.

This data distribution causes problems while training the MLP, so we tackled them with weighted sampling and data augmentation.

Weighted sampling is unequally sampling from classes to reduce the distribution bias. Our experiments showed that sampling the green data points 134% more than the red points gave the best performance. Data augmentation is increasing the amount of data to help the training. In our case, we said it is evident that regions with high confidence detection or tracking scores must give high confidence in the next time step. So we added new green data points at scores higher than 0.94, which is the maximum value in the generated dataset. Our experiments showed that adding 1000 points enhances the performance. Figure 7.9 shows the training data after applying the weighted sampling and data augmentation. For the rest of the experiments, we will use these techniques.

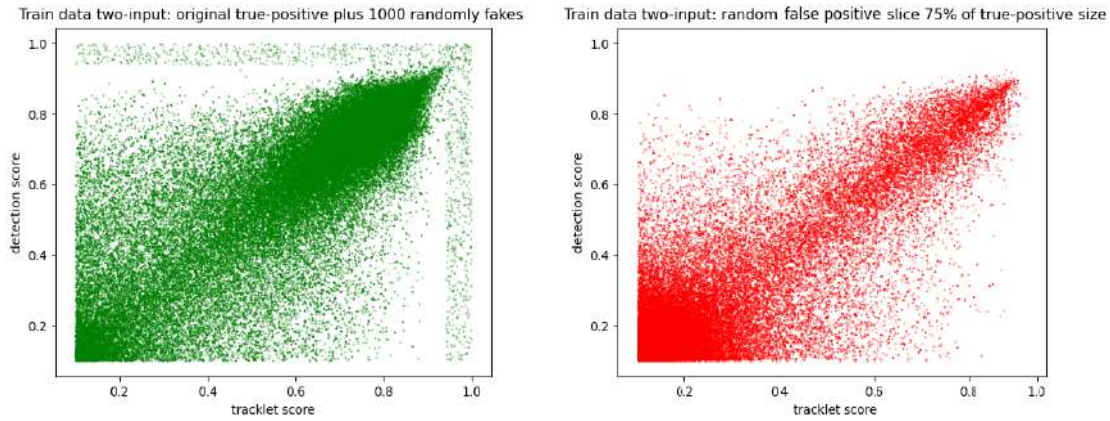


Figure 7.9: The training data for the MLP score update function after applying the weighted sampling and data augmentation. The figure on the left with the green points are the data points that are labeled as one, and the red points on the right are labeled as zero.

Now that we have our training data, we start training the MLP. We fixed the learning rate at 0.01 and used Stochastic gradient descent (SGD) with a momentum value of 0.9 as

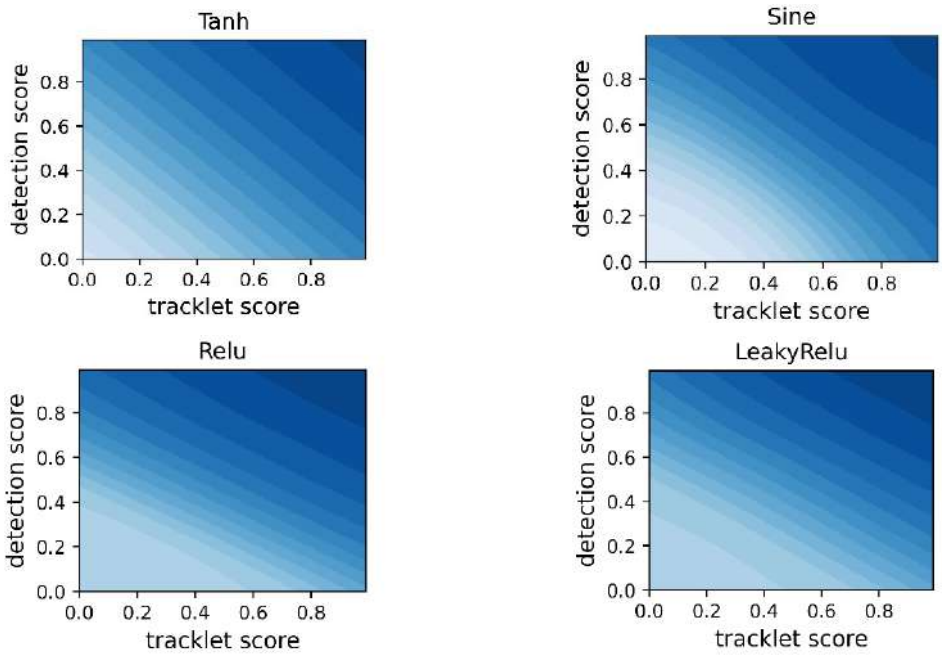


Figure 7.10: Contour plot of the different update function outputs using: a) Tanh, b) Sine, c) ReLU, and d) Leaky ReLU

an optimizer. We did a hyperparameter search, and we found that the optimized number of layers and hidden units are 4 and 100, respectively. Furthermore, we did experiments to find the best activation function. We chose the most popular activation functions: Relu, Leaky Relu, Tanh, and Sine. Table 7.4 shows the results of the validation split, and the best results will be evaluated on the test split to check the generalization.

We see that all the tracking systems that used the MLP for score refinement gave higher AMOTA from the previous work (Yin *et al.*, 2020). The best result with an AMOTA of **69.4** was using leaky ReLU, where it increased the AMOTA by **2.2**. In addition to the results, we wanted to see the embedded function that the networks had learned. We evaluated the networks at different detection and tracklet scores and plotted their output as shown in figure 7.10. We can see that the functions that the networks learned are simple, smooth, and behave like a weighted sum. The activation functions that gave high results (leaky ReLU and ReLU) give a higher weight to the detection score than the tracking score.

7.3.5 Class dependent MLP

We also used a three-input MLP, which contains, besides the two scores, an index of the class of the object. We noticed a significant unbalance between the number of points for each class during data generation. This can lead to bias in training, leading to inaccurate

results. Therefore, we performed weighted sampling to sample from each class equally. We chose the best results in table 7.4and used its hyperparameters. After evaluating the network, we got an AMOTA of 65.3 and MOTA of 53.5, which are poor compared to our previous results.

7.3.6 Dynamic max-distance threshold

When we changed the max-distance threshold, the parameter used for matching the detections and tracklets, we significantly improved AMOTA (section 7.3.3). So we decided to experiment with a max-distance threshold that depends on the object's state. Is it moving or not? We think the system must be more confident when an object is not moving. That is why we think that the max-distance threshold of a moving object should not be the same as a stationary object. So we reduced the max-distance threshold of stationary objects by half.

State	AMOTA↑	MOTA ↑	FP↓	FN↑	IDS ↓
Before	69.4	59.6	13543	**18896**	505
After	**69.5**	**60.2**	**12865**	18937	**502**

Table 7.5: Result of dynamic max-distance threshold on our best result so far. evaluated on the val split

We have a slight improvement which we believe is a promising one. Still, this change is not too dynamic because it is the same reduction in all frames. We believe that the MOT matching will improve more when we get the max-distance threshold of each time step separately.

7.3.7 Late-fusion ensemble

We tested our score refinement and compared it to the voting ensemble methods. As a baseline, we used our score refined CenterPoint as a LIDAR-based tracker, and a corrected version of CenterTrack (Zhou *et al.*, 2020a) as a camera-based tracker. Unfortunately, CenterTrack appears to produce an error in its tracking ids. Remanaging its ids (ensuring the id's uniqueness) significantly affected its performance. We achieved an AMOTA of 17.75 on the nuScenes validation set, in contrast to its reported AMOTA of 6.8 (Zhou *et al.*, 2020a), making it, in fact, one of the best camera-based trackers on the nuScenes data set at that time. In our previous experiments, we applied score decay only on tracklets, not on new detections. Whereas with the fusion of two modalities, it seems more reasonable to apply score decay equally. Thus, we tested by employing a *score-decay* of 0.2 to either of the modalities or both.

Method	LIDAR	Cam.	AMOTA↑	MOTA↑	FP↓	FN↓	IDS↓
No fusion	✓	-	67.51	**58.3**	13666	**18882**	494
No fusion	-	✓	17.75	15.04	15783	55009	7576
Affirmative	✓	✓	67.35	52.65	18445	23194	360
Consensus	✓	✓	50.80	48.73	**8980**	31573	659
Score refinement	σ_{score}	✓	67.33	52.86	18060	25485	**353**
	✓	σ_{score}	68.37	54.27	17874	20265	442
	σ_{score}	σ_{score}	68.73	54.48	15988	23235	529
	σ_{score}*	σ_{score}*	**69.18**	56.30	17661	20304	459

Table 7.6: Tracking results with different ensemble methods. σ_{score} Score-decay applied * only if tracklets are unmatched

Table 7.6 shows the results of late-fusion ensemble experiment. Here, we did not use the optimized max-distance thresholds. We see the improvement of confidence-based ensemble methods with equal treatment of both modalities against other ensemble methods. We achieve the most significant improvement of 1.67 in the AMOTA score against the LIDAR-based tracker, if we treat both modalities equally and only apply a *score-decay* if both modalities disagree. The other ensemble methods did not improve our baseline. While these strategies count the number of agreements between modalities, they do not account for confidence differences.

7.3.8 Combining everything

In the last section, we did not use the optimized max-distance thresholds and the MLP score update function. In this section, we want to combine everything we have developed so far. We used late fusion between the Lidar and the camera detector, optimized max-distance thresholds, and the different score update functions. Table 7.7 shows a summary of the experiments we have done so far and shows the contribution of the individual components. We see that the overall improvement from the baseline is 6.4 AMOTA points, which is a significant improvement. We see that the most significant contribution is the late fusion, with an increase of 1.7 AMOTA points. The least significant one is using an MLP as an update function with an increase of 0.2 AMOTA points and a decrease of 0.6 in MOTA points.

7.4 Comparison with the state of the art methods

To compare our method against the state-of-the-art methods, we ran our methods on the test split using the nuScenes challenge evaluation server. We compared our best LIDAR and multi-modal configurations to the state-of-the-art trackers on the nuScenes

Update function	OMD	Mod.	AMOTA↑	MOTA ↑	FP↓	FN↓	IDS ↓
-	-	L	65.9	56	13317	20274	562
-	✓	L	67.1	57.5	**12771**	20254	544
Eq. 7.6	-	L	67.5	58.3	13666	**18882**	494
Eq. 7.6	-	L + C	69.2	56.3	17661	20304	459
Eq. 7.6	✓	L + C	72.1	**58.5**	17473	19701	466
MLP	✓	L + C	**72.3**	57.9	18121	20128	**441**

Table 7.7: The results of our method with different components on the val split of nuScenes. OMD, Mod, L and C stand for Optimized Maximum Distance threshold, Modality, LIDAR and camera, respectively.

Method	LIDAR	Cam.	AMOTA↑	MOTA↑	FP↓	FN↓	IDS↓
(Yin *et al.*, 2020)	✓	-	65.0	54.5	16469	24092	557
Tracker*	✓	-	65.6	54.3	16631	24116	732
MCMOT*	✓	✓	66.6	55.6	16322	23065	1803
(Kim *et al.*, 2021)	✓	✓	67.7	56.8	17705	24925	1156
Octopus Tracker*	✓	-	67.9	57.2	16970	22272	781
AlphaTrack*	✓	-	**69.3**	57.6	18421	22996	718
CBMOT (ours)	✓	-	68.0	**58.3**	**15715**	**21217**	**548**
CBMOT (ours)	✓	✓	68.3	55.4	21585	21221	661

Table 7.8: Comparative evaluation on nuScenes test split. * unpublished work

test evaluation, and table 7.8 shows the results. We see that our approach is not far from the best performing method and was the best-published work. Our Lidar tracker has the best MOTA, FP, FN, and IDS. We won the NuScenes tracking challenge of 2021 with our method. Moreover, the good results on the test split show that the MLP did not over-fit the training data and was able to generalize to the test data.

7.5 Conclusion

This work showed that *confidence-based* MOT works better than *count-based* MOT when using *score-decay* and a proper score update function. We made an ablation study to see the effects of the hyperparameters and score update functions. We found out that we can neglect most hyperparameters but the score-decay. In addition, we found that using an MLP is the best score update function. Our approach consistently showed improvements when using it with different detectors, filtering algorithms, and datasets. We also revealed that optimizing the max-distance threshold could affect the performance

significantly, and this optimization played a significant role in our improvement. We also proposed a dynamic max-distance threshold which we believe will improve the performance further. Furthermore, we used score refinement as a late-fusion ensemble method in a multi-modal pipeline. It reinforced the score of matched tracklets from different modalities, resulting in an improvement of 1.67 in AMOTA. We demonstrated that the overall AMOTA improvement is 6.4 points compared to the baseline for the Lidar-camera tracker with the MLP score update function resulting in an AMOTA score of 68.3 on the nuScenes test set. We showed that our method is comparable to state-of-the-art methods, got second place on the leaderboard, and won the nuScenes tracking challenge. Furthermore, our LIDAR tracker had the best MOTA, FN, FP, and IDS. Finally, we believe that our approach is general and can be applied to even further tasks, such as 2D MOT.

Chapter 8

Overall Conclusion

8.1 Summary

In this dissertation, we saw how important autonomous driving is and how it could save lives, provide mobility, reduce wasted time driving, and enable new ways to design our cities. We focused on the bottleneck of autonomous driving, which is perception and particularly the instance segmentation and 3D multi-object tracking tasks.

In chapter 3 based on Riaz *et al.* (2020), we designed FourierNet, which is a single-stage anchor-free method for instance segmentation. We used a novel training technique with IFFT as a differentiable shape decoder that decodes the mask into a cartesian or a polar representation. We showed empirically that the Cartesian representation has a higher upper bound than the polar representation; however, it was hard to train and gave a lower performance in experiments. We believe that the reason for that is that the polar representation is a 1D problem, which is easier to solve than the 2D cartesian optimization problem.

Moreover, we could obtain a compact mask representation with low frequencies because they contain most of the mask?s information. Therefore, FourierNet outperformed all methods which use less than 20 parameters quantitatively and qualitatively. Furthermore, compared to object detectors, FourierNet can yield better approximations of objects than boxes, using slightly more parameters. Furthermore, we showed that our normalized centerness is generally a better centerness metric than the polar centerness when we do not use the centerness factor. Finally, our FourierNet-640 achieves a real-time speed of 26.6 FPS, and the FourierNet with ResNext-101 achieved comparable results to other polygon and implicit representations.

After that in chapter 4 based on Benbarka *et al.* (2022), we showed that a Fourier mapped perceptron with an integer lattice mapping is precisely the d-dimensional Fourier series. As a result, one perceptron with a large enough lattice can represent any signal. We demonstrated experimentally on the image regression task that one perceptron with frequencies equal to the Nyquist rate of the whole image could reconstruct it perfectly. Furthermore, we showed that our modified progressive training strategy of adding sequentially low to high frequencies worked on arbitrary mappings and improved the generalization of the interpolation task. In addition, we showed that a Fourier mapped per-

ceptron is structurally like a one hidden layer SIREN but with a trainable mapping. We saw that trainable mappings help if we train with a low number of frequencies and hurt when we use a high number of frequencies. Furthermore, in Fourier mapped MLPs, we showed that the integer lattice mapping forces the neural network?s underlying function periodicity. Lastly, we confirmed experimentally on the image regression and novel view synthesis tasks that the main contributor to reconstruction performance using a Fourier mapped MLP is the size of its mapping and the standard deviation of its elements.

In chapter 5 based on Riaz *et al.* (2021), we showed how implicit representations combined with the Fourier series can be applied to the task of instance segmentation to generate high-quality masks. We illustrate that the masks generated using our Fourier mapping are compact and meaningful. The lower Fourier frequencies hold the shape and higher frequencies hold the sharp edges. Furthermore, by sub-sampling the pixel coordinates in our implicit MLP, we can generate higher resolution masks during inference, which are visually smoother and improve the mAP over our baseline Mask R-CNN with similar settings and model capacity.

In chapter 7 based on Benbarka *et al.* (2021) and Gherri (2021), we showed that confidence-based MOT works better than count-based MOT when using score-decay and a proper score update function. We made an ablation study to see the effects of the hyperparameters and score update functions. We found out that we can neglect most hyperparameters but the score-decay. In addition, we found that using an MLP is the best score update function. Our approach consistently showed improvements when using it with different detectors, filtering algorithms, and datasets. We also revealed that optimizing the max-distance threshold could affect the performance significantly, and this optimization played a significant role in our improvement. We also proposed a dynamic max-distance threshold which we believe will improve the performance further.

Furthermore, we used score refinement as a late-fusion ensemble method in a multi-modal pipeline. It reinforced the score of matched tracklets from different modalities, resulting in an improvement of 1.67 in AMOTA. We demonstrated that the overall AMOTA improvement is 6.4 points compared to the baseline for the Lidar-camera tracker with the MLP score update function resulting in an AMOTA score of 68.3 on the nuScenes test set. We showed that our method is comparable to state-of-the-art methods, got second place on the leaderboard, and won the nuScenes tracking challenge. Furthermore, our LIDAR tracker had the best MOTA, FN, FP, and IDS. Finally, we believe that our approach is general and can be applied to even further tasks, such as 2D MOT.

Abbreviations

ANN	Artificial Neural Network
AMOTA	Average Multi Object Tracking Accuracy
BARF	Bundle-Adjusting Neural Radiance Fields
CF	Centerness Factor
CNN	Convolutional Neural Networks
COCO	Common Objects in COntext
CR	Coefficient Regression
DARPA	Defense Advanced Research Projects Agency
DSD	differentiable shape decoder
FCOS	Fully Convolutional One-Stage object detector
FFT	Fast Fourier Transform
FNN	Fourier Neural Networks
GC	Gaussian Centerness
GPS	Global Positional System
HTC	Hybrid Task Cascade
IFFT	Inverse Fast Fourier Transform
ILSVRC	ImageNet Large Scale Visual Recognition Challenge
IMU	Inertial inertial measurement units
INR	Implicit neural representation
IoU	Intersection-over-Union
KITTI	Karlsruhe Institute of Technology and Toyota Technological Institute
LIDAR	LIght Detection And Ranging
mAP	mean Average Precision
MLP	Multi-Layer Perceptron
MRI	Magnetic resonance imaging
NC	Normalized Centerness
NeRF	Neural Radiance Fields
NTK	Neural Tangent Kernel
NVS	Novel View Synthesis
PC	Polar Centerness
PE	Positional Encoding
PSNR	Peak signal-to-noise ratio
PT	Progressive Training

RADAR	RAdio Detection And Ranging
ReLU	Rectified Linear Unit
RoI	Region-of-Interest
RPN	Region Proposal Network
SIREN	SInusoidal REpresentation Networks
SONAR	SOund Detection And Ranging
WI	Weight Initialization
YOLACT	You Only Look At CoefficienTs

Bibliography

Alison McClelland, F. M. (1998). The social consequences of unemployment. https://library.bsl.org.au/jspui/bitstream/1/266/1/social_consequences_of_unemployment_AMcClelland.pdf.

Atzmon, M. and Lipman, Y. (2020). Sal: Sign agnostic learning of shapes from raw data. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2565–2574.

Bai, M. and Urtasun, R. (2017). Deep watershed transform for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 5221–5229.

Baser, E., Balasubramanian, V., Bhattacharyya, P., and Czarnecki, K. (2019). Fantrack: 3d multi-object tracking with feature association network. In *2019 IEEE Intelligent Vehicles Symposium (IV)*. IEEE.

Behringer, R., Sundareswaran, s., Gregory, B., Elsley, R., Addison, B., Guthmiller, W., Daily, R., and Bevly, D. (2004). The darpa grand challenge - development of an autonomous vehicle. pages 226 – 231.

Bergmann, P., Meinhardt, T., and Leal-Taixe, L. (2019). Tracking without bells and whistles. In *Proceedings of the IEEE international conference on computer vision*, pages 941–951.

Bewley, A., Ge, Z., Ott, L., Ramos, F., and Upcroft, B. (2016). Simple online and realtime tracking. In *2016 IEEE International Conference on Image Processing (ICIP)*, pages 3464–3468. IEEE.

Bolya, D., Zhou, C., Xiao, F., and Lee, Y. J. (2019). Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9157–9166.

Caesar, H., Bankiti, V., Lang, A. H., Vora, S., Liong, V. E., Xu, Q., Krishnan, A., Pan, Y., Baldan, G., and Beijbom, O. (2020). nuscenes: A multimodal dataset for autonomous driving. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11621–11631.

Casado-Garcıa, A. and Heras, J. (2020). Ensemble methods for object detection. In *Proceedings of the ECAI European Conference on Artificial Intelligence*, pages 2688–2695.

Chabra, R., Lenssen, J. E., Ilg, E., Schmidt, T., Straub, J., Lovegrove, S., and Newcombe, R. (2020). Deep local shapes: Learning local sdf priors for detailed 3d reconstruction. In *European Conference on Computer Vision*, pages 608–625. Springer.

Chang, M.-F., Lambert, J., Sangkloy, P., Singh, J., Bak, S., Hartnett, A., Wang, D., Carr, P., Lucey, S., Ramanan, D., *et al.* (2019). Argoverse: 3d tracking and forecasting with rich maps. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8748–8757.

Chen, K., Pang, J., Wang, J., Xiong, Y., Li, X., Sun, S., Feng, W., Liu, Z., Shi, J., Ouyang, W., *et al.* (2019). Hybrid task cascade for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4974–4983.

Chen, Z. (2019). *IM-NET: Learning implicit fields for generative shape modeling*. Ph.D. thesis, Applied Sciences: School of Computing Science.

Chen, Z. and Zhang, H. (2019). Learning implicit fields for generative shape modeling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5939–5948.

Cheng, S., Leng, Z., *et al.* (2020). Improving 3d object detection through progressive population based augmentation. *arXiv preprint arXiv:2004.00831*.

Chiu, H.-k., Prioletti, A., Li, J., and Bohg, J. (2020). Probabilistic 3d multi-object tracking for autonomous driving. *arXiv preprint arXiv:2001.05673*.

De Brabandere, B., Neven, D., and Van Gool, L. (2017). Semantic instance segmentation with a discriminative loss function. *arXiv preprint arXiv:1708.02551*.

Deng, B., Lewis, J. P., Jeruzalski, T., Pons-Moll, G., Hinton, G., Norouzi, M., and Tagliasacchi, A. (2020). Nasa neural articulated shape approximation. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 612–628. Springer.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., and Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*, pages 248–255. Ieee.

Dickmanns, E. D., Behringer, R., Dickmanns, D., Hildebrandt, T., Maurer, M., Thomanek, F., and Schiehlen, J. (1994). The seeing passenger car'vamors-p'. In *Proceedings of the Intelligent Vehicles' 94 Symposium*, pages 68–73. IEEE.

Engelking, C. (2017). The 'driverless' car era began more than 90 years ago. https://www.discovermagazine.com/technology/the-driverless-car-era-began-more-than-90-years-ago.

Fan, H., Su, H., and Guibas, L. J. (2017). A point set generation network for 3d object reconstruction from a single image. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 605–613.

Gallant, A. R. and White, H. (1988). There exists a neural network that does not make avoidable mistakes. In *ICNN*, pages 657–664.

Gao, N., Shan, Y., Wang, Y., Zhao, X., Yu, Y., Yang, M., and Huang, K. (2019). Ssap: Single-shot instance segmentation with affinity pyramid. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 642–651.

Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving? the kitti vision benchmark suite. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 3354–3361. IEEE.

Genova, K., Cole, F., Vlasic, D., Sarna, A., Freeman, W. T., and Funkhouser, T. (2019). Learning shape templates with structured implicit functions. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 7154–7164.

Genova, K., Cole, F., Sud, A., Sarna, A., and Funkhouser, T. (2020). Local deep implicit functions for 3d shape. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4857–4866.

Gherri, A. (2021). Learning score update functions for confidence-based mot.

Gropp, A., Yariv, L., Haim, N., Atzmon, M., and Lipman, Y. (2020). Implicit geometric regularization for learning shapes. In *International Conference on Machine Learning*, pages 3789–3799. PMLR.

He, K., Zhang, X., Ren, S., and Sun, J. (2015). Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778.

He, K., Gkioxari, G., Dollár, P., and Girshick, R. (2017). Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969.

Henzler, P., Mitra, N. J., and Ritschel, T. (2020). Learning a neural 3d texture space from 2d exemplars. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 8356–8364.

Höfer, T., Shamsafar, F., Benbarka, N., and Zell, A. (2021). Object detection and autoencoder-based 6d pose estimation for highly cluttered bin picking. In *2021 IEEE International Conference on Image Processing (ICIP)*, pages 704–708. IEEE.

Houston, J., Zuidhof, G., Bergamini, L., Ye, Y., Chen, L., Jain, A., Omari, S., Iglovikov, V., and Ondruska, P. (2021). One thousand and one hours: Self-driving motion prediction dataset. In *Conference on Robot Learning*, pages 409–418. PMLR.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

Hu, P., Ziglar, J., Held, D., and Ramanan, D. (2020). What you see is what you get: Exploiting visibility for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 11001–11009.

Huang, Z., Huang, L., Gong, Y., Huang, C., and Wang, X. (2019). Mask scoring r-cnn. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6409–6418.

Jacot, A., Hongler, C., and Gabriel, F. (2018). Neural tangent kernel: Convergence and generalization in neural networks. In *NeurIPS*.

Janai, J., Güney, F., Behl, A., and Geiger, A. (2017). Computer vision for autonomous vehicles: Problems, datasets and state of the art. *Arxiv*, pages arXiv–1704.

Jiang, C., Sud, A., Makadia, A., Huang, J., Nießner, M., Funkhouser, T., *et al.* (2020). Local implicit grid representations for 3d scenes. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6001–6010.

Jochem, T., Pomerleau, D., Kumar, B., and Armstrong, J. (1995). Pans: A portable navigation platform. In *Proceedings of the Intelligent Vehicles' 95. Symposium*, pages 107–112. IEEE.

Kalman, R. E. *et al.* (1960). A new approach to linear filtering and prediction problems. *Journal of basic Engineering*, **82**, 35–45.

Kim, A., Ošep, A., and Leal-Taixé, L. (2021). Eagermot: Real-time 3d multi-object tracking and segmentation via sensor fusion. In *IEEE International Conference on Robotics and Automation (ICRA)*, pages 11315–11321.

Kirillov, A., Levinkov, E., Andres, B., Savchynskyy, B., and Rother, C. (2017). Instancecut: from edges to instances with multicut. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5008–5017.

Kirillov, A., He, K., Girshick, R., Rother, C., and Dollár, P. (2019). Panoptic segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9404–9413.

Kirillov, A., Wu, Y., He, K., and Girshick, R. (2020). Pointrend: Image segmentation as rendering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

KRA (2014). Futurama at 1939 ny world's fair. `https://www.youtube.com/watch?v=sClZqfnWqmc`.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, **25**, 1097–1105.

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

Kumar, H. (2019). Technical fridays. `https://kharshit.github.io/blog/2019/09/20/evaluation-metrics-for-object-detection-and-segmentation`.

Kuo, W., Angelova, A., Malik, J., and Lin, T.-Y. (2019). Shapemask: Learning to segment novel objects by refining shape priors. *arXiv preprint arXiv:1904.03239*.

Lang, A. H., Vora, S., Caesar, H., Zhou, L., Yang, J., and Beijbom, O. (2019). Pointpillars: Fast encoders for object detection from point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 12697–12705.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, **86**(11), 2278–2324.

Lee, Y. and Park, J. (2019). Centermask: Real-time anchor-free instance segmentation. *arXiv preprint arXiv:1911.06667*.

Liang, J., Homayounfar, N., Ma, W.-C., Xiong, Y., Hu, R., and Urtasun, R. (2019). Polytransform: Deep polygon transformer for instance segmentation. *arXiv preprint arXiv:1912.02801*.

Lin, C.-H., Ma, W.-C., Torralba, A., and Lucey, S. (2021). Barf: Bundle-adjusting neural radiance fields. *arXiv preprint arXiv:2104.06405*.

Lin, T.-Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., and Zitnick, C. L. (2014). Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer.

Lin, T.-Y., Dollár, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017a). Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125.

Lin, T.-Y., Goyal, P., Girshick, R., He, K., and Dollár, P. (2017b). Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988.

Liu, S. (2013). Fourier neural network for machine learning. In *2013 International Conference on Machine Learning and Cybernetics*, volume 1, pages 285–290. IEEE.

Liu, S., Qi, L., Qin, H., Shi, J., and Jia, J. (2018). Path aggregation network for instance segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8759–8768.

Liu, S., Saito, S., Chen, W., and Li, H. (2019). Learning to infer implicit surfaces without 3d supervision. *Advances in Neural Information Processing Systems*, **32**, 8295–8306.

Liu, S., Zhang, Y., Peng, S., Shi, B., Pollefeys, M., and Cui, Z. (2020). Dist: Rendering deep implicit signed distance function with differentiable sphere tracing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 2019–2028.

Liu, Z., Lin, Y., Cao, Y., Hu, H., Wei, Y., Zhang, Z., Lin, S., and Guo, B. (2021). Swin transformer: Hierarchical vision transformer using shifted windows. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 10012–10022.

Mahalanobis, P. C. (1936). On the generalized distance in statistics. National Institute of Science of India.

Mao, J., Niu, M., Jiang, C., Liang, H., Chen, J., Liang, X., Li, Y., Ye, C., Zhang, W., Li, Z., *et al.* (2021). One million scenes for autonomous driving: Once dataset. *arXiv preprint arXiv:2106.11037*.

Maurer, M., Behringer, R., Furst, S., Thomanek, F., and Dickmanns, E. D. (1996). A compact vision system for road vehicle guidance. In *Proceedings of 13th International Conference on Pattern Recognition*, volume 3, pages 313–317. IEEE.

MBG (2022). The first automobile. `https://group.mercedes-benz.com/company/tradition/company-history/1885-1886.html`.

Mescheder, L., Oechsle, M., Niemeyer, M., Nowozin, S., and Geiger, A. (2019). Occupancy networks: Learning 3d reconstruction in function space. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4460–4470.

Michalkiewicz, M., Pontes, J. K., Jack, D., Baktashmotlagh, M., and Eriksson, A. (2019). Implicit surface representations as layers in neural networks. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4743–4752.

Mildenhall, B., Srinivasan, P. P., Tancik, M., Barron, J. T., Ramamoorthi, R., and Ng, R. (2020). Nerf: Representing scenes as neural radiance fields for view synthesis. In *European Conference on Computer Vision*, pages 405–421. Springer.

NHTSA (2019). Traffic safety facts annual report. `https://www-fars.nhtsa.dot.gov/Main/index.aspx`.

Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. (2019). Occupancy flow: 4d reconstruction by learning particle dynamics. In *International Conference on Computer Vision*.

Niemeyer, M., Mescheder, L., Oechsle, M., and Geiger, A. (2020). Differentiable volumetric rendering: Learning implicit 3d representations without 3d supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 3504–3515.

Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., and Geiger, A. (2019a). Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540.

Oechsle, M., Mescheder, L., Niemeyer, M., Strauss, T., and Geiger, A. (2019b). Texture fields: Learning texture representations in function space. In *International Conference on Computer Vision*.

Osgood, B. G. (2019). *Lectures on the Fourier transform and its applications*, volume 33. American Mathematical Soc.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019). Deepsdf: Learning continuous signed distance functions for shape representation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 165–174.

Peng, S., Niemeyer, M., Mescheder, L., Pollefeys, M., and Geiger, A. (2020). Convolutional occupancy networks. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part III 16*, pages 523–540. Springer.

Qi, C. R., Su, H., Mo, K., and Guibas, L. J. (2017). Pointnet: Deep learning on point sets for 3d classification and segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 652–660.

Ratti, C. (2021). Singapore is shaping the future of mobility. https://www.project-syndicate.org/commentary/singapore-mobility-experiments-driverless-cars-by-carlo-ratti-2021-0

Ren, S., He, K., Girshick, R., and Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99.

Riaz, M., Benbarka, N., Zell, A., *et al.* (2020). Fouriernet: Compact mask representation for instance segmentation using differentiable shape decoders. *arXiv e-prints*, pages arXiv–2002.

Riaz, M., Benbarka, N., Hoeffer, T., Zell, A., *et al.* (2021). Fouriermask: Instance segmentation using fourier mapping in implicit neural networks. *arXiv e-prints*, pages arXiv–2112.

Saito, S., Huang, Z., Natsume, R., Morishima, S., Kanazawa, A., and Li, H. (2019). Pifu: Pixel-aligned implicit function for high-resolution clothed human digitization. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 2304–2314.

Schulter, S., Vernaza, P., Choi, W., and Chandraker, M. (2017). Deep network flow for multi-object tracking. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6951–6960.

Shi, S., Wang, X., and Li, H. (2019a). Pointrcnn: 3d object proposal generation and detection from point cloud. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–779.

Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., and Li, H. (2019b). Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. *arXiv preprint arXiv:1912.13192*.

Shi, S., Guo, C., Jiang, L., Wang, Z., Shi, J., Wang, X., and Li, H. (2020). Pv-rcnn: Point-voxel feature set abstraction for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10529–10538.

Silvescu, A. (1999). Fourier neural networks. In *IJCNN'99. International Joint Conference on Neural Networks. Proceedings (Cat. No. 99CH36339)*, volume 1, pages 488–491. IEEE.

Simonyan, K. and Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

Sitzmann, V., Zollhöfer, M., and Wetzstein, G. (2019). Scene representation networks: continuous 3d-structure-aware neural scene representations. In *Proceedings of the 33rd International Conference on Neural Information Processing Systems*, pages 1121–1132.

Sitzmann, V., Martel, J., Bergman, A., Lindell, D., and Wetzstein, G. (2020). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, **33**.

Stanley, K. O. (2007). Compositional pattern producing networks: A novel abstraction of development. *Genetic programming and evolvable machines*, **8**(2), 131–162.

Sun, P., Kretzschmar, H., Dotiwalla, X., Chouard, A., Patnaik, V., Tsui, P., Guo, J., Zhou, Y., Chai, Y., Caine, B., *et al.* (2019). Scalability in perception for autonomous driving: An open dataset benchmark. *arXiv preprint arXiv:1912.04838*.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., Erhan, D., Vanhoucke, V., and Rabinovich, A. (2015). Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9.

Szegedy, C., Ioffe, S., Vanhoucke, V., and Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning. In *Thirty-first AAAI conference on artificial intelligence*.

Tan, M. and Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks. In *International conference on machine learning*, pages 6105–6114. PMLR.

Tancik, M., Srinivasan, P. P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J. T., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*.

Thrun, S., Montemerlo, M., Dahlkamp, H., Stavens, D., Aron, A., Diebel, J., Fong, P., Gale, J., Halpenny, M., Hoffmann, G., *et al.* (2006). Stanley: The robot that won the darpa grand challenge. *Journal of field Robotics*, **23**(9), 661–692.

Tian, Z., Shen, C., Chen, H., and He, T. (2019). Fcos: Fully convolutional one-stage object detection. *arXiv preprint arXiv:1904.01355*.

Tuttle, D., Stone, P., Beeson, P., Mericli, T., and Madigan, R. (2007). Darpa urban challenge technical report.

Benbarka, N., Schröder, J., and Zell, A. (2021). Score refinement for confidence-based 3d multi-object tracking. In *2021 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 8083–8090. IEEE.

Benbarka, N., Höfer, T., Zell, A., *et al.* (2022). Seeing implicit neural representations as fourier series. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2041–2050.

Varga, L. A., Messmer, M., Benbarka, N., and Zell, A. (2023). Wavelength-aware 2d convolutions for hyperspectral imaging. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*.

Vora, S., Lang, A. H., Helou, B., and Beijbom, O. (2020). Pointpainting: Sequential fusion for 3d object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4604–4612.

VPRO (2019). Are self-driving cars the future? `https://www.youtube.com/watch?v=5gyxjWERSU8`.

Weng, X. and Kitani, K. (2019). A baseline for 3d multi-object tracking. *arXiv preprint arXiv:1907.03961*.

Weng, X., Wang, Y., Man, Y., and Kitani, K. M. (2020). Gnn3dmot: Graph neural network for 3d multi-object tracking with 2d-3d multi-feature learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6499–6508.

WHO, W. h. o. (2018). Global status report on road safety 2018. `https://www.who.int/news-room/fact-sheets/detail/road-traffic-injuries`.

Wojke, N., Bewley, A., and Paulus, D. (2017). Simple online and realtime tracking with a deep association metric. In *2017 IEEE international conference on image processing (ICIP)*, pages 3645–3649. IEEE.

Wu, Y., Kirillov, A., Massa, F., Lo, W.-Y., and Girshick, R. (2019). Detectron2. `https://github.com/facebookresearch/detectron2`.

Xie, E., Sun, P., Song, X., Wang, W., Liu, X., Liang, D., Shen, C., and Luo, P. (2019). Polarmask: Single shot instance segmentation with polar representation. *arXiv preprint arXiv:1909.13226*.

Xie, E., Sun, P., Song, X., Wang, W., Liu, X., Liang, D., Shen, C., and Luo, P. (2020). Polarmask: Single shot instance segmentation with polar representation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 12193–12202.

Xie, S., Girshick, R., Dollár, P., Tu, Z., and He, K. (2017). Aggregated residual transformations for deep neural networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1492–1500.

Xu, W., Wang, H., Qi, F., and Lu, C. (2019). Explicit shape encoding for real-time instance segmentation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5168–5177.

Yan, Y., Mao, Y., and Li, B. (2018). Second: Sparsely embedded convolutional detection. *Sensors*, **18**(10), 3337.

Yang, B., Luo, W., and Urtasun, R. (2018). Pixor: Real-time 3d object detection from point clouds. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 7652–7660.

Yang, Z., Xu, Y., Xue, H., Zhang, Z., Urtasun, R., Wang, L., Lin, S., and Hu, H. (2019). Dense reppoints: Representing visual objects with dense point sets. *arXiv preprint arXiv:1912.11473*.

Yin, T., Zhou, X., and Krähenbühl, P. (2020). Center-based 3d object detection and tracking. *arXiv:2006.11275*.

Ying, H., Huang, Z., Liu, S., Shao, T., and Zhou, K. (2019). Embedmask: Embedding coupling for one-stage instance segmentation. *arXiv preprint arXiv:1912.01954*.

Yu, J., Jiang, Y., Wang, Z., Cao, Z., and Huang, T. (2016). Unitbox: An advanced object detection network. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 516–520.

Zhou, B., Krähenbühl, P., and Koltun, V. (2019a). Does computer vision matter for action? *Science Robotics*, **4**(30), eaaw6661.

Zhou, X., Zhuo, J., and Krahenbuhl, P. (2019b). Bottom-up object detection by grouping extreme and center points. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 850–859.

Zhou, X., Wang, D., and Krähenbühl, P. (2019c). Objects as points. *arXiv:1904.07850*.

Zhou, X., Koltun, V., and Krähenbühl, P. (2020a). Tracking objects as points. *arXiv:2004.01177*.

Zhou, Y. and Tuzel, O. (2018). Voxelnet: End-to-end learning for point cloud based 3d object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4490–4499.

Zhou, Y., Sun, P., Zhang, Y., Anguelov, D., Gao, J., Ouyang, T., Guo, J., Ngiam, J., and Vasudevan, V. (2020b). End-to-end multi-view fusion for 3d object detection in lidar point clouds. In *Conference on Robot Learning*, pages 923–932.

Zhu, B., Jiang, Z., Zhou, X., Li, Z., and Yu, G. (2019). Class-balanced grouping and sampling for point cloud 3d object detection. *arXiv:1908.09492*.