Implicit Object Pose Estimation on RGB Images Using Deep Learning Methods

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. Timon Höfer

aus Tübingen

Tübingen

2023

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

Tag der mündlichen Qualifikation:	29.09.2023
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr. Andreas Zell
2. Berichterstatter/-in:	Prof. Dr. Hendrik Lensch

Eine Sache vollkommen zu verstehen und dieselbe Sache vollkommen mißzuverstehen, sind zwei Dinge, die nicht immer vollständig voneinander getrennt werden können.

Robert Musil

Abstract

With the increasing availability of robotic and camera systems, as well as the success of deep learning based methods in computer vision, it is of interest to determine not only the position of objects, but also their exact orientation. This enables, among other things, automated bin picking where an attached camera sensor uses image data or point clouds to determine the orientation of the objects at hand, allowing an attendant robotic arm to use this information to perform a successful grasp. This is certainly not the only reason to do pose recognition. In autonomous driving, among other things, the orientation of a car can be used to predict its trajectory, or in the field of virtual reality, virtual objects have to be transformed depending on the person's point of view.

The focus of this dissertation is on implicit RGB-based methods. While methods based on depth data have been common solutions in the past, they have, among other drawbacks, the high initial cost of a depth camera; moreover, depth data are often not precise enough for small objects. Developing methods based only on RGB data, however, is the more difficult problem. In addition to localization, one has to determine the distance and also the orientation, which seems almost impossible with conventional methods. In recent years, however, researchers have managed to do just that - using Deep Learning, it was now possible to predict poses of objects on RGB images. Still, typical problems get in the way. For example the overlapping of objects by other elements, different light effects, which have influence on the appearance of the object, or existing symmetries of the objects.

For this purpose, we introduce two implicit Deep Learning based methods for pose estimation on RGB images. Among other things, we present a complete process from data generation to selection of the best poses. The advantage of focusing on implicit methods can be seen as follows. In explicit methods, it should be noted that conventional parameterizations of rotations, such as the Euler angles, are not continuous in their representation, from which pose regression has to suffer. Furthermore, in case of existing symmetries, it is typically necessary to perform an additional annotation of these. Such problems can be elegantly circumvented by an implicitly given pose determination, as we will see in the following.

Kurzfassung

Mit der zunehmenden Verfügbarkeit an Roboter- und Kamerasystemen, sowie dem Erfolg Deep Learning basierter Methoden im Bereich Computer Vision ist es von Interesse, neben der Position von Objekten auch die genaue Orientierung dieser zu bestimmen. Dies ermöglicht unter anderem den automatisierten Griff in die Kiste, bei der ein angebrachter Kamerasensor über Bilddaten oder Punktwolken die Ausrichtung der vorliegenden Objekte bestimmt, sodass ein beistehender Roboterarm mithilfe dieser Information einen erfolgreichen Griff durchführen kann. Dies ist sicherlich nicht der einzige Grund um Posenerkennung zu betreiben, im autonomen Fahren kann unter anderem durch die Ausrichtung eines Autos dessen Trajektorie vorausgesagt werden, oder im Bereich der virtuellen Realität müssen virtuelle Objekte je nach Blickwinkel der Person umgewandelt werden.

Der Fokus dieser Dissertation liegt dabei auf impliziten, RGB-basierenden Methoden. Während Methoden, die auf Tiefendaten basieren, in der Vergangenheit eine gängige Lösung darstellten, haben sie unter anderem die Nachteile der hohen Anschaffungskosten einer Tiefenkamera, zudem sind die Tiefendaten bei kleinen Objekten oft nicht präzise genug. Methoden, die nur auf RGB Daten basieren, zu entwickeln ist das schwierigere Problem. Neben der Lokalisation muss man die Entfernung und auch die Orientierung ermitteln, was mit herkömmlichen Methoden nahezu unmöglich erscheint. In den letzten Jahren gelang es Forschern aber eben genau dies zu schaffen - unter Gebrauch von Deep Learning war es nun möglich, Posen von Objekten auf RGB-Bildern vorherzusagen. Trotzdem stellen sich typische Probleme in den Weg. Zum einen, die Überdeckung von Objekten durch andere Elemente, unterschiedliche Lichteinwirkungen, welche Einfluss auf die Erscheinung des Objektes haben, oder vorhandene Symmetrien der Objekte.

Hierzu führen wir zwei implizite, auf Deep Learning basierende, Methoden für die Posenerkennung auf RGB Bildern ein. Wir stellen unter anderem einen kompletten Prozess von der Datengenerierung bis hin zur Selektion der besten Posen vor. Der Vorteil, der sich dadurch ergibt, dass wir uns auf implizite Methoden fokussieren, kann wie folgt gesehen werden: In expliziten Methoden ist zu beachten, dass herkömmliche parametrisierungen der Rotationen, wie zum Beispiel der eulersche Winkel, nicht stetig in ihrer Repräsentation sind, worunter eine Posenregression leiden muss. Zudem ist es typischerweise notwendig, im Falle von vorhandenen Symmetrien eine zusätzliche Annotation dieser durchzuführen. Durch eine implizit gegebene Posenbestimmung können solche Probleme elegant umgangen werden, wie wir im Folgenden sehen werden.

Contents

Chapter 1

Introduction

1.1 Motivation

Sight, hearing, smell, taste and touch - these are the 5 senses of a human. ”The human is an eye animal,” it is often said. In fact, the sense of sight is considered the most important sense for our conscious perception. This is one of the reasons why it is much easier for us to describe what we see and put it into words than, for example, a smell. The importance of the sense of sight in everyday life is obvious: In most daily activities, sighted people rely in some way on the functioning of their visual system. This starts with reading this text and applies in a similar way to many activities - from shopping and watching movies to driving and playing soccer. The sense of sight performs important services when it comes to perceiving our surroundings and orienting ourselves in space, that is: when it comes to locating objects and determining their size or their distances from each other, but also when we perform movements.

With the ubiquity of camera sensors, the question arises whether we are able to give the robot a visual sense. The camera sensors are supposed to play the role of the eye, and our neural networks with their thousands of neurons are supposed to play the nervous system so that the robot is able to process the sensory stimuli it receives. This gives the robot the ability to communicate with the environment so that it can move in rooms without bumping into things or grasp things.

In the field of computer vision, visual processing is considered. Problems such as understanding underlying geometry, distance and position of things, and classifying objects are analyzed. Object recognition includes the task of object identification in images and videos. Among others, it includes the task of image classification, object detection and segmentation.

Especially the task of image classification was a driving force behind the evolution of Deep Learning in computer vision. Krizhevsky *et al.* (2012) revolutionized the recognition field and computer vision in general with their deep-learning based AlexNet submission to the ImageNet Challenge (by Russakovsky *et al.* (2015)), which took place from 2012 to 2015. As a result of the improvement in performance of AlexNet, neural networks, in particular convolutional neural networks (CNNs), have become a part of most state-of-the-art object recognition algorithms.

Specific to robotics applications is the task of object recognition. Here, the task is to localize and classify objects in images up to pixel-wise segmentation. Knowledge about the location and presence of objects is essential for the successful execution of robot control. Notable detectors in this regard are the R-CNN series, which, starting from the region-based detector R-CNN by Girshick *et al.* (2014), and improved versions, such as Fast R-CNN by Girshick (2015) and Faster R-CNN by Ren *et al.* (2015), and for simultaneous segmentation Mask R-CNN by He *et al.* (2017). In contrast, there is the series of YOLO detectors, whose name ranges from YOLO version 1 by Redmon *et al.* (2016) to YOLO version 7, which is convincing in speed as a one-stage method.

In order to actually use such detectors, one needs, in addition to the special hardware, such as the GPU, large data sets on which one can train and evaluate the models. Generating such datasets is very laborious, from capturing images in different scenarios to image-by-image labeling of visible objects on the images. Publicly available datasets, such as ImageNet by Deng *et al.* (2009) with several million images or COCO by Lin *et al.* (2014) and PascalVOC by Everingham *et al.* (2010a) enable the application and development of deep learning-based methods in computer vision. In the industrial sector, it is also possible to generate synthetic data with, e.g. BlenderProc by Denninger *et al.* (2019) for industrial or household objects with a given CAD model. Through augmentations and a physics engine it is possible to train models in the synthetic world and still get good results in the real world.

Now suppose the robot recognizes the position and class of an object. It is possible for it to move to this object. To actually interact with this object, e.g. to pick it up, more knowledge about the object is necessary. Here, the object pose plays a role, which gives information about the orientation and position of the object in a three-dimensional space.

Traditionally, pose estimation can be handled by template matching as done, e.g. by Hinterstoisser *et al.* (2012). In recent years, however, there has been a rise of deep learning-based methods that have beaten the classical methods in terms of speed and precision. Among other things, previously hand-crafted features have now been made learnable, or pose regression has been applied directly. In particular, the handling of the difficulties of pose determination, including occlusion, clutter, illumination changes, textureless objects and symmetries, improved. The BOP challenge by Hodaň *et al.* (2020) should be mentioned in this context. Starting in 2019, it took place annually and will look specifically at pose estimation datasets that contain the above difficulties. It is a challenge for researchers on 6D pose estimation on multiple well-known 6D datasets, such as YCB by Xiang *et al.* (2017), containing textureless and symmetric objects, strong occlusions and clutter. Over time, RGB-based methods could prove themselves over depth-based methods. If one gives them the possibility of a pose refinement, for example by the ICP (iterative closest point) algorithm, they are not only faster but also achieve a higher precision. Note that the required depth data can be obtained from an RGB-D camera. However, symmetries and occlusions in particular are still a problem. For example, two viewpoints can look identical on the image but have different pose labels. Learning the average of the poses would again be error-prone, so typically, symmetry annotations of

Figure 1.1: In the image, a Franka Emika robot arm equipped with a Schmalz cobot suction gripper is shown in action, moving purposefully towards its target. Upon arrival, the suction gripper deftly picks up the object of interest with precision guided by our pose estimation method. This technique leverages the input of the Microsoft Azure Kinect RGB-D camera, concealed behind a sturdy aluminum mount, which captures a detailed image of the scene to provide crucial information about the object's position and rotation.

the objects are included in the training data so that this problem can be partially avoided. Finding a solution that can handle this problem well is a crucial component of a robot vision pipeline.

In this dissertation, we will highlight a method for synthetic data generation for industrial objects with a given CAD model based on BlenderProc and will demonstrate that we can use it for training and testing our detectors and pose estimators. Hereby we bridge the gap between simulation and reality using further augmentations. Furthermore, we present two implicit methods on pose estimation that cover the uncertainty, including the removal of the need for symmetry supervision, in a clever way. The first one is based on an underlying implicit autoencoder network. Latent representations of a discrete subset from SO(3) will be saved via a codebook, and during inference, it will be evaluated, which of the rotations matches the image crop of the bounding box the most. In Figure 1.1 a successful grasp based on our pose estimation framework is shown. The other method we introduce is based on implicit neural representations, which acts as a probability distribution on SO(3). We will show that in case of multiple poses due to symmetry, our method HyperPosePDF is able to capture all of them. The detailed contributions are listed below.

1.2 Contribution & Outline

This thesis contributes to the computer vision and robotics communities with the following peer-reviewed works, where * indicates equal contribution:

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

   In this work, we focus on the Fourier embedding used for implicit neural representations. Motivated by a mathematical analysis, we introduce an integer Fourier mapping that is equivalent to the original Fourier series for a perceptron and show that it forces periodicity of the network output. Furthermore, we explore the mathematical connection between Fourier mappings and SIRENs and show that a Fourier-mapped perceptron is structurally like a one hidden layer SIREN. Finally, we confirm that the main contributor to the performance is the number of elements and the standard deviation of the Fourier mapping.

2. Timon Höfer and Andreas Zell. **"Automatic Adjustment of Fourier Embeddings."** In Proceedings of the IEEE International Conference on Pattern Recognition (ICPR) 2022.

   In this work, we introduce an iterative adjustment method for the Fourier embedding of implicit neural representations. We compare two pruning techniques in selecting which elements of the Fourier embedding are the most unimportant and introduce a replacement technique that is chosen in such a way as to have a positive influence on the overall standard deviation. We can demonstrate that an initial poorly chosen Fourier embedding can be adjusted with our iterative replacement method to yield adequate performance.

3. Riaz, Hamd Ul Moqeet*, Nuri Benbarka*, Timon Höfer, and Andreas Zell. **"FourierMask: Instance Segmentation using Fourier Mapping in Implicit Neural Networks."** In Proceedings of the International Conference: Image Analysis and Processing (ICIAP) 2022.

   Here, we focus on the task of object segmentation. We provide a mask representation using a Fourier embedding and implicit neural representations to represent the level set. We show that this mask representation outperforms the widely spread grid-based masks. While this method uses Mask R-CNN as a baseline, we are able to keep up a similar speed but produce more accurate output.

4. Timon Höfer, Faranak Shamsafar, Nuri Benbarka and Andreas Zell. **"Object detection and Autoencoder-Based 6D Pose Estimation for Highly Cluttered Bin Picking."** In Proceedings of the IEEE International Conference on Image Processing (ICIP) 2021.

We present a framework for pose estimation in highly cluttered bin picking scenarios where we assume the CAD model of the respective objects to be given and an RGB-D camera installed on top of the bin. We start by creating a synthetic dataset to remove the need for expensive hand labelling of the pose annotations. We combine a state-of-the-art object detector with an Autoencoder network to estimate the poses. Furthermore, we present a pose filtering scheme to select the best pose predictions. The usage of a denoising autoencoder isolates pose information and hence makes symmetry supervision unnecessary.

5. Timon Höfer, Benjamin Kiefer and Andreas Zell. **"HyperPosePDF: Prediciting the Probability Distribution on SO(3)."** In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV) 2023.

   We present HyperPosePDF, a method to predict the probability distribution on SO(3). An initial vision network takes the image as input and outputs the weights of an MLP, which represents the implicit neural representation of the pose probability function. This is especially helpful for symmetries, as the network naturally inherits any present ambiguities.

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

6. Benjamin Kiefer, ... , Timon Höfer et al.. **"1st Workshop on Maritime Computer Vision (MaCVi) 2023: Challenge Results."** In Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision Workshops (WACVW) 2023.

   The 1st MaCVi Workshop 2023 focuses on computer vision in maritime environments for UAVs and USVs. The workshop consists of four subchallenges: (i) UAV-based maritime object detection, (ii) UAV-based maritime object tracking, (iii) USV-based maritime obstacle segmentation, and (iv) USV-based maritime obstacle detection, based on SeaDronesSee and MODS benchmarks. The report summarizes the subchallenges' main findings and introduces a new benchmark, SeaDronesSee Object Detection v2, which includes more classes and footage. The report offers statistical and qualitative analyses and assesses trends in over 130 submissions' best-performing methodologies.

7. Benjamin Kiefer, Timon Höfer and Andreas Zell. **"Stable Yaw Estimation of Boats from the Viewpoint of UAVs and USVs."** (Under review)

   In this paper, we propose a method for yaw estimation of boats from the perspective of UAVs and USVs. Several applications rely on it, such as 3D scene rendering, trajectory prediction, and navigation. This paper addresses the lack of literature on the topic and extends HyperPosePDF to handle video-based scenarios. As demonstrated in our experimental evaluation, aggregating probability distributions of pose predictions improves performance. This method has potential benefits for downstream tasks in marine robotics.

As the last two works are unrelated to the main topic, it is not going to be discussed in this dissertation. The structure of the thesis is as follows:

Chapter 2 In the first chapter, we lay out the fundamentals of the mathematical theory. We introduce rotations and different parameterizations that are commonly used for pose estimation. After a short discussion on their usability, we proceed with the introduction to the classical Fourier theory. This will be important for the introduction of Fourier embeddings which are crucial for implicit neural representations. Furthermore, we discuss the Fourier theory on the rotation manifold, as we will introduce in the last chapter an implicit neural representation that models a probability distribution on SO(3).

Chapter 3 Next, we introduce implicit neural representations, which use multilayer perceptrons to represent high-frequency functions. We show that utilizing the theory on the Fourier series does increase the performance significantly. In this chapter, we present our papers 'Seeing implicit neural representations as Fourier series' by Benbarka & Höfer *et al.* (2022) and 'Automatic adjustment of Fourier embedding parameterizations' by Höfer and Zell (2022). Both works focus on the Fourier embedding, while in this section, we focus on the task of image regression and novel view synthesis. We will later see how to use them for the task of instance segmentation and pose estimation.

Chapter 4 This chapter presents the tasks of object detection and object segmentation. We show how implicit neural representations can be used for the task of object segmentation in our work 'FourierMask: Instance segmentation using Fourier mapping in implicit neural networks' by Riaz *et al.* (2022).

Chapter 5 In this chapter, we introduce our work 'Object detection and Autoencoder-based 6D pose estimation for highly cluttered industrial bin picking by Höfer *et al.* (2021), which presents a full framework for pose estimation in highly cluttered bin picking scenarios. We combine the task of object detection and instance segmentation with pose estimation, which makes it possible to predict industrial objects in heavily cluttered scenarios.

Chapter 6 In the second to last chapter, we present our work 'HyperPosePDF: Predicting the probability distribution on SO(3)' by Höfer *et al.* (2023), which predicts a probability distribution on SO(3) and is especially well suited for modelling uncertainties arising from, e.g. symmetry. This work uses implicit neural representations as its implicit pose probability function and outputs in the presence of symmetries multiple or even continuous poses.

Chapter 7 The last chapter gives a short overall conclusion of the work presented in this thesis.

Chapter 2

Mathematical Theory

In this section, we want to derive the essential formulas used in the remainder of the work. Since our goal is to use machine learning for pose estimation, we first introduce different rotation parameterizations and then discuss their effect on the pose regression task. A key element of implicit neural representations is an initial Fourier embedding, which will be discussed in Section 3. Here we lay out the basics and introduce the Fourier series for common functions and the Fourier series on SO(3). With that, we have the fundamentals to introduce a hypernetwork that predicts an implicit neural representation of a pose probability function on SO(3).

2.1 Rotation Representation

Rotations are used to describe the orientation of objects. For example, in the task of bin picking, not only the 3D location of an object is needed for a successful grasp, rather the whole 6D pose is necessary to determine optimal grasping points for the robot arm. Several rotation representations have been proposed for the task of pose estimation. The well-known Euler angles representation does not perform well in the regression task, as it suffers from singularities. Some works make use of the quaternion representation, which is free of singularities. Still, we introduce the Euler angles as we need them as the parameterization for the Fourier embedding on SO(3). This gives us the opportunity to simultaneously discuss the drawbacks and see the advantages of quaternions over the Euler angle parameterization. We will now continue with introducing rotation matrices.

2.1.1 Three Dimensional Rotations

We start the section with an intuitive definition of three dimensional rotations. We will mainly follow Murray *et al.* (2017) and Prestin (2010) for our introduction to rotations.

Definition 2.1.1 *A rotation acting on* $\mathbb{R}^3$ *around the origin* $\mathbf{0}$ *is a linear map* $\rho : \mathbb{R}^3 \to \mathbb{R}^3$*, such that* $\rho(\mathbf{v}) = \mathbf{R}\mathbf{v}$ *with the following properties*

- $\mathbf{R} \in \mathbb{R}^{3x3}$ *is an orthogonal matrix, i.e.* $\mathbf{R}^\mathbf{T}\mathbf{R} = \mathbf{I}$.

- *It holds that* $\det(\mathbf{R}) = 1$*, i.e. the rotation matrix is norm preserving.*

We collect basic properties of the so defined rotations in the following Lemma.

Remark 2.1.2 *Let* $\rho_1(\mathbf{v}) = \mathbf{R_1 v}$ *and* $\rho_2(\mathbf{v}) = \mathbf{R_2 v}$ *be two rotations according to the previous definition. It holds that*

1. *The composition of two rotations* $\rho = \rho_2 \circ \rho_1$ *is the map* $\rho : \mathbf{v} \mapsto \mathbf{R}_2\mathbf{R}_1\mathbf{v}$.
2. *The inverse* ρ^{-1} *of a rotations is the map* $\rho^{-1} : \mathbf{v} \mapsto \mathbf{R}^{-1}\mathbf{v}$.
3. *The equivalence* $\mathbf{R}_1 \neq \mathbf{R}_2 \Leftrightarrow \rho_1 \neq \rho_2$ *holds.*

This remark gives a one-to-one correspondence between a rotation ρ and its associated rotation matrix $\mathbf{R}$. Hence, we will identify the two from here on and refer to the rotation matrix $\mathbf{R}$.

Lemma 2.1.3 *The set* $\mathscr{R} = \{\mathbf{R} \in \mathbb{R}^{3\times3} | \det(\mathbf{R}) = 1$ *and* $\mathbf{R}^T\mathbf{R} = \mathbf{I}\}$ *associated with matrix multiplication forms the group* $(\mathscr{R}, \cdot)$.

Proof: We will prove the validity of the group axioms:

1. We show that $\mathbf{R}_1, \mathbf{R}_2 \in \mathscr{R} \Rightarrow \mathbf{R}_1\mathbf{R}_2 \in \mathscr{R}$.
   - Per definition of the matrix mulitiplication we have that $\mathbf{R}_1\mathbf{R}_2 \in \mathbb{R}^{3\times3}$
   - Multiplicativity of the determinant yields $\det(\mathbf{R}_1\mathbf{R}_2) = \det(\mathbf{R}_1)\det(\mathbf{R}_2) = 1$.
   - The product is orthogonal as can be seen by $(\mathbf{R}_1\mathbf{R}_2)^T(\mathbf{R}_1\mathbf{R}_2) = \mathbf{R}_2^T(\mathbf{R}_1^T\mathbf{R}_1)\mathbf{R}_2 = \mathbf{R}_2^T\mathbf{R}_2 = \mathbf{I}$.
2. The associativity comes from the fact that the matrix multiplication is associative: $(\mathbf{R}_1\mathbf{R}_2)\mathbf{R}_3 = \mathbf{R}_1(\mathbf{R}_2\mathbf{R}_3)$.
3. The identity element is the identity matrix of the matrix multiplication $\mathbf{I}$ for which it holds that $\mathbf{IR} = \mathbf{R}$, $\forall \mathbf{R} \in \mathscr{R}$. As $\mathbf{I}$ fulfills the properties of $\mathscr{R}$ it follows that $\mathbf{I} \in \mathscr{R}$ which is needed.
4. The existence of the inverse is guaranteed as for $\mathbf{R} \in \mathscr{R}$ it holds that $\det(\mathbf{R}^{-1}) = \det(\mathbf{R})^{-1} = 1$ and $(\mathbf{R}^{-1})^T\mathbf{R}^{-1} = (\mathbf{R}\mathbf{R}^T)^{-1} = \mathbf{I}$. Hence it holds that $\mathbf{R}^{-1} \in \mathscr{R}$ and as $\mathbf{R}$ is orthogonal we have $\mathbf{R}^{-1} = \mathbf{R}^T$.

□

Definition 2.1.4 *We call the group* $(\mathscr{R}, \cdot)$ *the orthogonal group* $\mathrm{SO}(3)$.

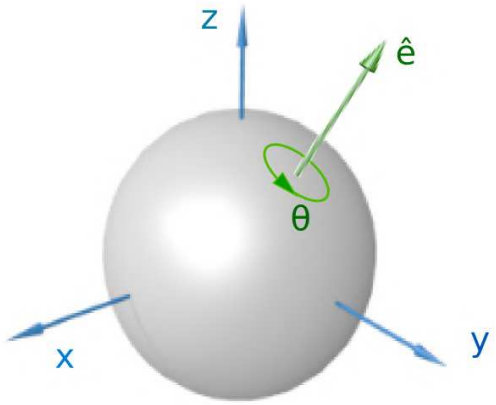


Figure 2.1: 3D visualization of a sphere and a rotation about an axis $\hat{\mathbf{e}}$ by an angle of θ, following the axis-angle parameterization 2.1.2 (Malan (2004)).

This definition is well known in the literature, and the group $\mathrm{SO}(3)$ is usually called the (3D-) rotation group. It should be mentioned that the rotation group is non-abelian, i.e. rotating something 90 degrees along one axis and then 90 degrees along another axis is not the same as doing them in reverse order. In general, a rotation matrix $\mathbf{R}$ has nine entries but is a constraint with limitations. If we write $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3]$ where $\mathbf{r}_i \in \mathbb{R}^3$ for $i = 1,2,3$, we can see the following: By definition $\mathbf{R}$ is orthogonal and hence we have $\mathbf{r}_i \cdot \mathbf{r}_j = \delta_{i,j}$ for $i, j = 1,2,3$. With the inner product being commutative, we end up with six constraints for the entries of $\mathbf{R}$. This indeed reduces the number of freely eligible entries from 9 to 3 for the rotation matrix $\mathbf{R}$. The condition to have a determinant of 1 does not affect the total number of freely eligible entries but only the number of possible choices. Based on this observation, we will use the expression of having three degrees of freedom in a rotation to refer to the three freely eligible elements.

For the task of pose estimation, we have nine parameters to be regressed for a single rotation, which is excessive compared to other parameterizations. When regressing these parameters with backpropagation, one has to enforce orthogonality such that the nine parameters actually represent a rotation matrix. Additionally, rotation matrices are not intuitive. By simply looking at a rotation matrix, it is usually not clear to say what rotation is represented, which makes other parameterizations more favourable for the task of pose regression.

2.1.2 Parameterizations of the Rotation

Axis-Angle Parameterization

The axis-angle parameterization is an intuitive way to describe a rotation. The description is given by the rotation angle θ around a rotation axis indicated by a unit vector $\mathbf{e}$. To define the direction of a unit vector e rooted at the origin, only two numbers are needed, not three, since the magnitude of $\mathbf{e}$ is constrained.

Definition 2.1.5 *Let $\mathbf{R} \in \mathrm{SO}(3)$ with $\mathbf{R} \neq \mathbf{I}$ be given. Then the **axis of rotation** is defined to be the normalized eigenvector $\mathbf{e}$ to the eigenvalue $\lambda = 1$ of $\mathbf{R}$.*

The exclusion of $\mathbf{R} = \mathbf{I}$ in the definition is due to the fact that $\mathbf{I}$ has a three-fold eigenvalue one and, therefore, no uniquely determined normalized eigenvector to this eigenvalue.

Lemma 2.1.6 *Let $\mathbf{R} = [\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3] \in \mathrm{SO}(3)$ with $\mathbf{R} \neq \mathbf{I}$ and $\mathbf{v} \in \mathbb{R}^3$ be given. Then the axis of rotation is given by*

$$\mathbf{e} = \frac{\mathbf{v}}{\|\mathbf{v}\|} \text{ with } \mathbf{v} = \begin{pmatrix} r_{2,3} - r_{3,2} \\ r_{3,1} - r_{1,3} \\ r_{1,2} - r_{2,1} \end{pmatrix}.$$

Proof: The orthogonality of $\mathbf{R}$ yields the following equivalence for an eigenvector $\mathbf{u}$ to $\lambda = 1$:

$$\begin{aligned} &\mathbf{R}\mathbf{u} = \mathbf{u} \overset{\text{orth.}}{\Leftrightarrow} \mathbf{u} = \mathbf{R}^T\mathbf{u} \\ \Leftrightarrow &(\mathbf{R} - \mathbf{R}^T)\mathbf{u} = 0 \\ \Leftrightarrow &\begin{pmatrix} 0 & (r_{1,2} - r_{2,1}) & (r_{1,3} - r_{3,1}) \\ (r_{2,1} - r_{1,2}) & 0 & (r_{2,3} - r_{3,2}) \\ (r_{3,1} - r_{1,3}) & (r_{3,2} - r_{2,3}) & 0 \end{pmatrix} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}. \end{aligned}$$

This is equivalent to the system of equations

$$\begin{aligned} u_2(r_{1,2} - r_{2,1}) - u_3(r_{3,1} - r_{1,3}) &= 0, \\ u_3(r_{2,3} - r_{3,2}) - u_1(r_{1,2} - r_{2,1}) &= 0, \\ u_1(r_{3,1} - r_{1,3}) - u_2(r_{2,3} - r_{3,2}) &= 0. \end{aligned}$$

The solution of these equations are multiples of $\mathbf{u} = (r_{2,3} - r_{3,2}, r_{3,1} - r_{1,3}, r_{1,2} - r_{2,1})$. Normalization by dividing through the norm makes the solution unique and proves our Lemma. □

Remark 2.1.7 *The two-dimensional unit sphere is defined as*

$$\mathbb{S}^2 = \{\mathbf{x} \in \mathbb{R}^3 \mid \|\mathbf{x}\| = 1\}.$$

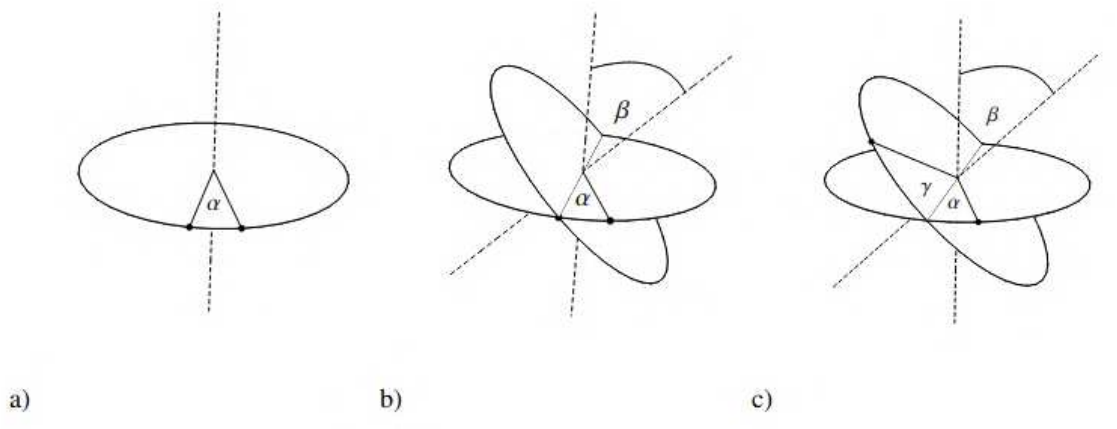


Figure 2.2: The Euler angles associated with the ZYZ-convention (Definition 2.1.10) are the consecutive rotations around the (a) z-axis, (b) y-axis and (c) the z-axis. Illustration taken from Prestin (2010).

Since we normalized the rotation axis $\mathbf{e}$*, we have* $\mathbf{e} \in \mathbb{S}^2$*. Hence, we can transfer knowledge from* $\mathbb{S}^2$ *to the set containing the rotation axes. Specifically, we know that each element* $\mathbf{x} = (x_1, x_2, x_3)^T \in \mathbb{S}^2$ *can be transformed in spherical coordinates* (φ, ψ) *where* $\varphi \in [0, 2\pi)$ *denotes the longitude and* $\psi \in [0, \pi]$ *denotes the latitude of the point on* $\mathbb{S}^2$*. If* $x_1 = x_2 = 0$ *the longitude is not uniquely determined, but otherwise, it holds that*

$$\varphi = \begin{cases} \arccos \frac{x_1}{\sqrt{x_1^2 + x_2^2}} & \text{for } x_2 \geq 0, \\ 2\pi - \arccos \frac{x_1}{\sqrt{x_1^2 + x_2^2}} & \text{for } x_2 < 0, \end{cases}$$

$$\psi = \arccos x_3.$$

Given φ *and* θ *it is also possible to calculate the Cartesian coordinates:*

$$\begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} \sin\psi \cos\varphi \\ \sin\psi \sin\varphi \\ \cos\psi \end{pmatrix}.$$

This remark shows that we need only two parameters to represent the rotation axis and hence only need three parameters to represent a rotation.

Euler Angle Parameterization

In the previously introduced axis-angle parameterization consisting of a rotation θ and a rotation axis $\mathbf{e}$, we showed that we only need three parameters, the rotation axis $\mathbf{e}$ represented by the spherical coordinates (φ, ψ) and the rotation angle ω. The Euler angle

parameterization uses a different approach to account for the three degrees of freedom. Specifically, we split the rotation into three rotations around the different axes and use their absolute value for characterization. We will need to define a convention for the Euler angles, and we will do so with regard to Section 2.3. In literature, there exist different conventions on choosing those axes, most commonly the ZXZ- and ZYZ-conventions. As both representations can be transformed into each other with given angles α, β, γ, by

$$\mathbf{R}_{ZYZ} = \mathbf{R}_{ZXZ}(\alpha + \pi/2, \beta, \gamma - \pi/2),$$

we introduce the ZYZ-convention.

Definition 2.1.8 *For* $\mathbf{e}_x = (1,0,0)^T$, $\mathbf{e}_y = (0,1,0)^T$ *and* $\mathbf{e}_z = (0,0,1)^T$ *we define the sets*

$$\begin{aligned}\mathscr{X} &= \{\mathbf{X} \in \mathrm{SO}(3) | \mathbf{X}\mathbf{e}_x = \mathbf{e}_x\},\\ \mathscr{Y} &= \{\mathbf{Y} \in \mathrm{SO}(3) | \mathbf{Y}\mathbf{e}_y = \mathbf{e}_y\},\\ \mathscr{Z} &= \{\mathbf{Z} \in \mathrm{SO}(3) | \mathbf{Z}\mathbf{e}_z = \mathbf{e}_z\}.\end{aligned}$$

Note that each set $\mathscr{X}$, $\mathscr{Y}$ *and* $\mathscr{Z}$ *is a subgroup of* $\mathrm{SO}(3)$ *with the property of being isomorphic to* $\mathrm{SO}(2)$.

Remark 2.1.9 *Every rotation* $\mathbf{R}_X, \mathbf{R}_Y, \mathbf{R}_Z \in \mathscr{X}, \mathscr{Y}, \mathscr{Z}$ *fulfills*

$$\mathbf{R}_X = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\gamma_x) & -\sin(\gamma_x) \\ 0 & \sin(\gamma_x) & \cos(\gamma_x) \end{pmatrix}, \quad \mathbf{R}_Y = \begin{pmatrix} \cos(\gamma_y) & 0 & \sin(\gamma_y) \\ 0 & 1 & 0 \\ -\sin(\gamma_y) & 0 & \cos(\gamma_y) \end{pmatrix},$$
$$\mathbf{R}_Z = \begin{pmatrix} \cos(\gamma_z) & -\sin(\gamma_z) & 0 \\ \sin(\gamma_z) & \cos(\gamma_z) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

for some $\gamma_x, \gamma_y, \gamma_z \in [0, 2\pi)$.

Definition 2.1.10 *Let* $\alpha, \gamma \in [0, 2\pi)$ *and* $\beta \in [0, \pi]$ *be given angles, then a rotation matrix is given by*

$$\mathbf{R}(\alpha, \beta, \gamma) = \mathbf{R}_Z(\alpha)\mathbf{R}_Y(\beta)\mathbf{R}_Z(\gamma).$$

This representation is called the ***Euler angle representation****, and* α, β *and* γ *are called the* ***Euler angles****.*

Remark 2.1.11 *Given the rotation matrix* $\mathbf{R} = (r_{i,j})_{i,j=1,2,3} \in \mathrm{SO}(3)$*, the corresponding Euler angles can be calculated as follows.*

If $|r_{3,3}| \neq 1$*, then we can calculate the Euler angles as*

$$\beta = \arccos r_{3,3}$$

$$\alpha = \begin{cases} \arccos \frac{r_{1,3}}{\sqrt{r_{1,3}^2 + r_{2,3}^2}} & \text{for } r_{2,3} > 0, \\ 2\pi - \arccos \frac{r_{1,3}}{\sqrt{r_{1,3}^2 + r_{2,3}^2}} & \text{for } r_{2,3} < 0, \end{cases}$$

$$\gamma = \begin{cases} \arccos \frac{-r_{3,1}}{\sqrt{r_{3,1}^2 + r_{3,2}^2}} & \text{for } r_{3,2} > 0, \\ 2\pi - \arccos \frac{-r_{3,1}}{\sqrt{r_{3,1}^2 + r_{3,2}^2}} & \text{for } r_{3,2} < 0. \end{cases}$$

In the case of $|r_{3,3}| = 1$ *it is*

$$\beta = 0$$

$$\alpha + \gamma = \begin{cases} \arccos r_{1,1} & \text{for } r_{2,1} \geq 0, \\ 2\pi - \arccos r_{1,1} & \text{for } r_{2,1} < 0. \end{cases}$$

In the case of $|r_{3,3}| = -1$ *it is*

$$\beta = \pi$$

$$\alpha + \gamma = \begin{cases} \arccos(-r_{1,1}) & \text{for } r_{2,1} \geq 0, \\ 2\pi - \arccos(-r_{1,1}) & \text{for } r_{2,1} < 0. \end{cases}$$

Note that every rotation matrix $\mathbf{R}$ *has uniquely determined Euler angles and, therefore, a unique Euler angle parameterization.*

A downside of Euler angles is that we can not rely on distances measured in the representation space, due to the GIMBAL LOCK, the loss of one degree of freedom that occurs when the axes of two of the three gimbals are driven into a parallel configuration, "locking" the system into rotation in a degenerate two-dimensional space. For further details see Figure 2.3 and Hemingway and O'Reilly (2018). Furthermore, rotations can be described in many ways with different sets of Euler angles, and hence we can not apply smoothing or averaging, which are important for animations. These shortcomings can be overcome with quaternions.

Quaternions to Represent Rotations

Quaternions avoid issues like gimbal lock and give a very seamless way to interpolate between two three-dimensional orientations, which lacks ambiguities of Euler angles and avoids the issues of numerical precision and normalization that arise in trying to

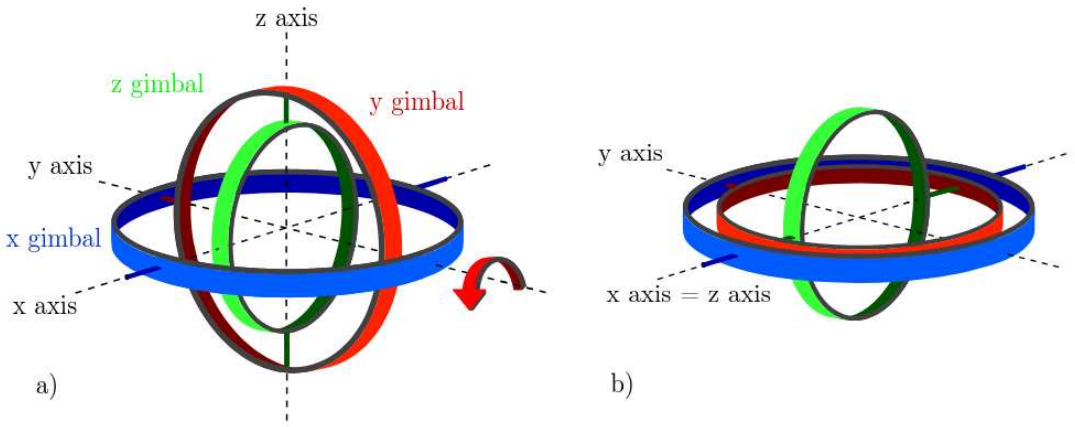


Figure 2.3: The occurrence of the gimbal lock as shown by Zeitlhöfler (2019): (a) The initial situation with three axes perpendicular to each other. (b) After rotating 90 degrees around the y axis, the x and z axes coincide.

interpolate between two rotation matrices. A quaternion is an expression of the form

$$q_0 + q_1 i + q_2 j + q_3 k,$$

where $q_0, q_1, q_2, q_3 \in \mathbb{R}$ and i, j, k are symbols that can be interpreted as unit-vectors pointing along the three spatial axes. They are complex valued and fulfill

$$i^2 = j^2 = k^2 = ijk = -1,$$

meaning that they are mutually orthogonal imaginary unit vectors. We denote $\mathbf{q} = (q_0, q_1, q_2, q_3)$ to specify a quaternion. We will restrict ourselves to a subset of quaternions that we call rotation quaternions. They are closely related to the previously introduced axis-angle representation.
With known axis-angle components $(\theta, \mathbf{e}) = (\theta, (e_0, e_1, e_2))$ it is possible to convert to rotation quaternions as follows:

$$\begin{aligned} q_0 &= \cos\left(\frac{\theta}{2}\right), \\ q_1 &= e_0 \sin\left(\frac{\theta}{2}\right), \\ q_2 &= e_1 \sin\left(\frac{\theta}{2}\right), \\ q_3 &= e_2 \sin\left(\frac{\theta}{2}\right). \end{aligned}$$

This equation shows that the real term (q_0) of the quaternion is completely determined by the rotation angle, while the imaginary terms (q_1, q_2 and q_3) are just the three rotation axis vectors scaled by a common factor. Therefore, a rotation quaternion's magnitude (that is, the sum of its four components squared) is always one.

It is reasonable to ask why we would bother with quaternions at all since axis-angle and quaternion representations both contain exactly the same information. We must perform these trigonometric operations anyway to do anything useful with an axis-angle quantity, such as rotate a set of points making up a 3D object. Performing them ahead of time allows most quaternion operations to be done with only multiplication/division and addition/subtraction, saving valuable computer resources.

Unlike the Euler angles, the quaternions are free from the gimbal lock problem, but they still have an ambiguity caused by their antipodal symmetry: $\mathbf{q}$ and $-\mathbf{q}$ correspond to the same rotation, which means that quaternions double cover the $\mathrm{SO}(3)$ group. One could try to avoid this ambiguity and restrict the quaternions to one hemisphere by restricting a scalar component to be positive, but as has been shown, similar orientations will still be far away in the representation.

Moreover, it was recently shown in Zhou *et al.* (2019b) that for 3D rotations, all representations are discontinuous in the real Euclidean spaces with four or fewer dimensions and empirical results suggest that continuous representations outperform discontinuous ones. Therefore, Euler angles, quaternions, exponential maps, and axis-angle representations might still not be optimal for regression. In our work we focus on implicit pose estimation algorithms, which considerably reduces the problem of selecting the optimal representation, as we will see later on.

2.2 General Fourier Theory

A key component of implicit neural representations that we will introduce in section 3 form Fourier embeddings. To understand the mathematics behind them, we will introduce them mathematically and see in the next section how we use them as an embedding to neural networks. In general, the Fourier transform decomposes functions depending on space or time into functions that are defined on spatial or temporal frequency. While the function itself will, in our case, be real-valued, the transform will be complex-valued to represent the complex sinusoids that comprise the original function.

2.2.1 The One-Dimenional Fourier transform

While there are different ways to define the Fourier transform for a complex-valued function $f : \mathbb{R} \to \mathbb{C}$, we introduce the integral representation for the definition. Furthermore, we will focus on real-valued functions $f : \mathbb{R} \to \mathbb{R}$ as this is our objective for the remaining chapters. Still, the results are also valid for complex functions.

Definition 2.2.1 *For a real-valued function $f : \mathbb{R} \to \mathbb{R}$, with $f \in L^1(\mathbb{R})$ we define the* ***Fourier transform*** *by*

$$\mathscr{F}[f](x) := \int_{-\infty}^{\infty} f(t)e^{-2\pi ixt}dt$$

for $x \in \mathbb{R}$.

While we are given a way to find the Fourier transform of a function, which we can conduct analyses on, we are also interested in the way back to the original function starting from the Fourier transform. A possibility to return to the function is given in the following theorem. If the conditions below are fulfilled, it is possible to recalculate the original function given the Fourier transform. These conditions are also known as the Dirichlet conditions.

Theorem 2.2.2 *Let $f : \mathbb{R} \to \mathbb{R}$ be a function that fulfils (1) $\int_{-\infty}^{\infty} |f|dt$ converges and (2) in any finite interval, f and its derivative f' are piecewise continuous with at most a finite number of maxima, minima and discontinuities. For $t \in \mathbb{R}$ where f is continuous we have*

$$f(t) = \int_{-\infty}^{\infty} \mathscr{F}[f](x)e^{2\pi itx}dx.$$

For $t \in \mathbb{R}$ where f is discontinuous we have

$$\frac{1}{2}[f(t+) + f(t-)] = \int_{-\infty}^{\infty} \mathscr{F}[f](x)e^{2\pi itx}dx,$$

with $f(t+)$ and $f(t-)$ denoting the right and left limits of f at t.

Proof: A proof can be found in Oppenheim *et al.* (1997). □

Theorem 2.2.3 *Let $f, g : \mathbb{R} \to \mathbb{R}$ be continuous and their derivatives f', g' be piecewise linear. If pointwise equality for the Fourier transform holds,*

$$\mathscr{F}[f](x) = \mathscr{F}[g](x), \quad \forall x \in \mathbb{R},$$

then the original functions are pointwise equal

$$f(t) = g(t) \quad \forall t \in \mathbb{R}.$$

Proof: This is an application of the Dirichlet theorem, as it holds that

$$\begin{aligned} f(t) &= \int_{-\infty}^{\infty} \mathscr{F}[f](x) e^{2\pi i t x} dx \\ &= \int_{-\infty}^{\infty} \mathscr{F}[g](x) e^{2\pi i t x} dx \\ &= g(t), \end{aligned}$$

for all $t \in \mathbb{R}$. □

This finding is crucial as it gives us a one-to-one correspondence of the Fourier transform and the original function, which enables analysis in the Fourier space. Contrary, if two functions would have the same Fourier transform, any analysis in the Fourier space would be redundant.

The Fourier Series

The main difference between the Fourier transform and the Fourier series is that the transform is applicable to signals that are not periodic, and the series is applicable to signals that are periodic. For the latter, we will assume the function f to be periodic on $[0,1]$. This is not a real constraint as if we are given a function f with period T, we could define the function $g(x) = f(Tt)$ with period 1.

Definition 2.2.4 *For a periodic function $f : \mathbb{R} \to \mathbb{R}$ on $[0,1]$ that is integrable, the **Fourier series** is the sum*

$$\sum_{k \in \mathbb{Z}} c_k e^{2\pi i k x},$$

*with the **Fourier coefficients** given as*

$$c_k = \int_0^1 f(t) e^{-2\pi i k t} dt.$$

Theorem 2.2.5 *For a function $f : \mathbb{R} \to \mathbb{C}$ that is periodic on $[0,1]$ and smooth, it holds that*

$$f(x) = \sum_{k \in \mathbb{Z}} c_k e^{2\pi i k x}.$$

Meaning that the Fourier sum represents the original function.

Proof: A proof can be found in Deitmar and Echterhoff (2014). □

As the definition of the Fourier coefficients is only done on an interval of finite length, the Fourier series itself is only defined for functions defined on an interval with finite

length, including periodic signals. An aperiodic signal can not be defined on an interval of finite length, and hence one must use the Fourier transform for such a signal. It would be possible to take the Fourier transform of a periodic signal by extending the function outside of $[0,1]$ with 0. The resulting Fourier transform would look like

$$\hat{f}(p) = \int_0^1 f(x)e^{-2\pi ipx}dx,$$

which is not particularly different from the Fourier coefficients. Furthermore, the ability to express a periodic signal as a discrete sum of frequencies is more meaningful than a continuous sum via the inversion formula.

2.2.2 The Multi-Dimensional Fourier Transform

The Fourier transform for multidimensional functions f defined on $\mathbb{R}^n$ follows the one-dimensional idea. We will denote by $\mathbf{x} = (x_1, \ldots, x_n)$ an n-dimensional vector with values in $\mathbb{R}^n$. The following definition is used for real- and complex-valued functions. In our application, we are looking at real-valued functions and therefore, we define the transform for real-valued functions.

Definition 2.2.6 *For a real valued function $f : \mathbb{R}^n \to \mathbb{R}$ the **multidimensional Fourier transform** is defined as*

$$\mathscr{F}[f](\xi) := \int_{\mathbb{R}^n} f(\mathbf{x})e^{-2\pi i\mathbf{x}\cdot\xi}d\mathbf{x}.$$

The dot product in the definition is defined as $\mathbf{x}\cdot\xi = \sum_{i=1}^n x_i\xi_i$ and acts as the key to extend the 1-dimensional Fourier transform for n-dimensional functions without affecting the appearance of its definition. The integral is over all of $\mathbb{R}^n$, and as an n-fold multiple integral, all the x_j's go from $-\infty$ to ∞. In coordinates, the Fourier transform reads as

$$\mathscr{F}[f]((\xi_1, \ldots, \xi_n)) := \int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} f((x_1, \ldots, x_n))e^{-2\pi i(x_1\xi_1 + \ldots + x_n\xi_n)}dx_1 \ldots dx_n.$$

As the coordinate-based notation is lengthy, especially for dimensions higher than 2, we will continue using the vector notation.

The multidimensional Fourier Series

Conveniently, most of the ideas and theorems for the Fourier series are also valid in the multidimensional case. Complex exponentials of a variable form the building blocks for periodic functions in the multidimensional case. In n-dimensions, we have the function

$$e^{2\pi ix_1}e^{2\pi ix_2}\ldots e^{2\pi ix_n}$$

that is, with period 1, periodic in each variable. While it is common in the 1-dimensional case to think of periodicity "in time", this way of thinking is not helpful in the multidimensional case, and hence we are not forcing ourselves to write the variable as "t". We question ourselves whether we can express a periodic function $f(x_1,\dots,x_n)$ as a Fourier series. We can answer this with "yes" and furthermore, the properties and formulas are like in the 1-dimensional case. For our work, it is enough to assume the period in each variable to be 1. With $\mathbf{m} = (m_1,\dots,m_n)$ we would imagine writing the Fourier series as

$$\sum_{\mathbf{m}\in\mathbb{Z}^n} c_{\mathbf{m}} e^{2\pi i m_1 x_1} e^{2\pi m_2 i x_2} \dots e^{2\pi i m_n x_n} = \sum_{\mathbf{m}\in\mathbb{Z}^n} c_{\mathbf{m}} e^{2\pi i \mathbf{m}\cdot\mathbf{x}},$$

with the sum over all integer combinations in $\mathbb{R}^n$. To find the coefficients $c_{\mathbf{m}}$ we extend the argument from the 1-dimensional case and find them to be given by

$$\begin{aligned}
&\int_0^1 \cdots \int_0^1 e^{-2\pi i m_1 x_1} e^{-2\pi m_2 i x_2} \dots e^{-2\pi i m_n x_n} f(x_1,\dots,x_n) dx_1 dx_2 \dots dx_n \\
&= \int_0^1 \cdots \int_0^1 e^{-2\pi i (m_1 x_1 + \cdots + m_n + x_n)} f(x_1,\dots,x_n) dx_1 \dots dx_n \\
&= \int_{[0,1]^n} e^{-2\pi i \mathbf{m}\cdot\mathbf{x}} f(x_1,\dots,x_n) d\mathbf{x}.
\end{aligned}$$

In summary, we find the Fourier series of a function $f(\mathbf{x})$ in $\mathbb{R}^n$ to be

$$\sum_{\mathbf{m}\in\mathbb{Z}^n} c_{\mathbf{m}} e^{2\pi i \mathbf{m}\cdot\mathbf{x}}, \tag{2.1}$$

with the Fourier coefficients given as

$$c_{\mathbf{m}} = \int_{[0,1]^n} e^{-2\pi i \mathbf{m}\cdot\mathbf{x}} f(x_1,\dots,x_n) d\mathbf{x}. \tag{2.2}$$

2.3 Fourier Series on the Rotation Manifold

Throughout this section we will use the ZYZ-convention of the Euler-angle parameterization, identifying an element $\mathbf{R} \in \mathrm{SO}(3)$ with $\mathbf{R}(\alpha,\beta,\gamma)$. For the sake of simplicity, we will also denote functions $f : \mathrm{SO}(3) \to \mathbb{C}$ by

$$f(\mathbf{R}(\alpha,\beta,\gamma)) = f(\alpha,\beta,\gamma).$$

2.3.1 A Basis for $L^2(SO(3))$

The Hilbert space $L^2(\mathrm{SO}(3))$ is associated with the inner product given by

$$\begin{aligned}\langle f_1, f_2\rangle &= \int_{\mathrm{SO}(3)} f_1(\mathbf{R})\overline{f_2(\mathbf{R})}d\mathbf{R} \\ &= \int_0^{2\pi}\int_0^{\pi}\int_0^{2\pi} f_1(\alpha,\beta,\gamma)\overline{f_2(\alpha,\beta,\gamma)}d\alpha d\beta d\gamma,\end{aligned}$$

for $f_1, f_2 \in L^2(\mathrm{SO}(3))$.

To define a basis system for $L^2(\mathrm{SO}(3))$ we introduce the Wigner-D and Wigner-d functions as they play a key role in Fourier analysis on $\mathrm{SO}(3)$. The Wigner-D functions $D_l^{m,n}(\mathbf{R})$ are the eigenfunctions of the Laplace operator for $\mathrm{SO}(3)$. The parameterization of these eigenfunctions in Euler angles yields an explicit expression for the Wigner-D functions, as shown by Chirikjian (2000),

$$D_l^{m,n}(\alpha,\beta,\gamma) = e^{-im\alpha}e^{-in\gamma}d_l^{m,n}(\cos(\beta)), \tag{2.3}$$

with $|m|,|n| \leq l \in \mathbb{N}_0$ and the Wigner-d functions being given as

$$d_l^{m,n}(x) = \frac{(-1)^{l-m}}{2^l}\sqrt{\frac{(l+m)!}{(l-n)!(l+n)!(l-m)!}}\cdot\sqrt{(1-x)^{n-m}}\frac{\mathrm{d}^{l-m}}{\mathrm{d}x^{l-m}}\frac{(1+x)^{n+l}}{(1-x)^{n-l}}. \tag{2.4}$$

The Peter-Weyl Theorem (Vilenkin (1978)) states that the harmonic spaces

$$\mathrm{Harm}_l(\mathrm{SO}(3)) = \mathrm{span}\{D_l^{m,n} : m,n = -l,\ldots,l\},$$

that are spanned by the Wigner-D functions satisfy, that the closure over their union yields $L^2(\mathrm{SO}(3))$:

$$L^2(\mathrm{SO}(3)) = \mathrm{clos}_{L^2}\bigoplus_{l=0}^{\infty}\mathrm{Harm}_l(\mathrm{SO}(3)). \tag{2.5}$$

This means that the set of Wigner-D functions $\{D_l^{m,n}(\mathbf{R}) : l \in \mathbb{N}_0,\ m,n = -l,\ldots,l\}$ forms an orthogonal basis system in $L^2(\mathrm{SO}(3))$. Hence we know about the existence of a representation for $f \in L^2\,\mathrm{SO}(3)$:

$$f(\mathbf{R}) = \sum_{l=0}^{\infty}\sum_{m=-l}^{l}\sum_{n=-l}^{l}\hat{f}_l^{m,n}D_l^{m,n}(\mathbf{R}) \tag{2.6}$$

Explicit coefficients are reported in Potts *et al.* (2007) and are given by the integral

$$\hat{f}_l^{m,n} = \frac{l+\frac{1}{2}}{4\pi^2}\langle f, D_l^{m,n}\rangle. \tag{2.7}$$

2.3.2 Discrete Fourier Transforms on $\mathrm{SO}(3)$

For $L \in \mathbb{N}$ we define the function spaces

$$\mathbb{D}_L = \bigoplus_{l=0}^{L} \mathrm{Harm}_l(\mathrm{SO}(3)), \tag{2.8}$$

with dimension

$$\dim(\mathbb{D}_L) = \sum_{l=0}^{L} (2l+1)^2 = \frac{1}{3}(L+1)(2L+1)(2L+3). \tag{2.9}$$

We assume the rotation to be given in Euler angles, and hence write $f(\alpha,\beta,\gamma)$ instead of $f(\mathbf{R}((\alpha,\beta,\gamma)))$. Restricting ourselves to L-band limited functions $f \in \mathbb{D}_L$ the Fourier sum reads as

$$\begin{aligned} f(\alpha,\beta,\gamma) &\overset{(2.6)}{=} \sum_{l=0}^{L} \sum_{m,n=-l}^{l} f_{l,m,n} D_l^{m,n}(\alpha,\beta,\gamma) \\ &\overset{(2.3)}{=} \sum_{l=0}^{L} \sum_{m,n=-l}^{l} f_{l,m,n} e^{-im\alpha} e^{-in\gamma} d_l^{m,n}(\cos(\beta)) \end{aligned}$$

Rearranging the sums yields

$$= \sum_{m=-L}^{L} e^{-im\alpha} \sum_{n=-L}^{L} e^{-in\gamma} \sum_{l=\max(|m|,|n|)}^{L} f_l^{m,n} d_l^{m,n}(\cos(\beta)).$$

To get rid of the sum on the right, we follow Potts *et al.* (2007) to transform a linear combination of the $d_l^{m,n}$'s into a linear combination of first kind Chebychev-polynomials which we call T_l. This results in

$$= \sum_{m,n=-L}^{L} e^{-im\alpha - in\gamma} \sum_{l=0}^{L} t_l^{m,n} T_l(\cos(\beta))(\sin(\beta)^{\mathrm{mod}(m+n,2)}.$$

We choose the coefficients $h_l^{m,n}$ such that they fulfil

$$\sum_{l=0}^{L} t_l^{m,n} T_l(\cos(\beta)) = \sum_{l=-L}^{L} h_l^{m,n} e^{-il\beta},$$

if $m+n$ is even and

$$\sin(\beta) \sum_{l=0}^{L} t_l^{m,n} T_l(\cos(\beta)) = \sum_{l=-L}^{L} h_l^{m,n} e^{-il\beta},$$

if $m+n$ is odd. Together, we receive

$$f(\alpha,\beta,\gamma) = \sum_{l,m,n=-L}^{L} h_l^{m,n} e^{-i\left((m,n,l)(R(\alpha,\beta,\gamma)\right)}. \tag{2.10}$$

Thus we see that any such function f can be written by a sum with coefficients $h_l^{m,n}$ multiplied with exponentials of rotations. In theory, we know that there should be a function that is able to map each rotation to the probability - that this is the underlying rotation in the current situation (e.g. on the current image). An analytical derivation of such a function is almost impossible. Especially, if symmetries are present that increase the complexity of this function. Therefore, the idea to approximate such a function by neural networks is close at hand. We will see later that it is possible to convert the exponential into a sinusoidal form. Practically, we can achieve this sinusoidal representation with a Fourier embedding on the input rotation. We do not have to calculate the coefficients $h_l^{m,n}$ by hand, but leave them learnable as weights of the neural network.

Chapter 3

Implicit Neural Representations

Implicit Neural Representations (INR) use multilayer perceptrons to represent high-frequency functions in low-dimensional problem domains. Recently, these representations achieved state-of-the-art results on tasks related to complex 3D objects and scenes. A core problem is the representation of highly detailed signals, which is tackled using networks with periodic activation functions (SIRENs) or applying Fourier mappings to the input. However, naively applying a Fourier embedding is a double-edged sword, and therefore we will analyze this in more detail within this chapter.

Therefore, in the first part of the chapter, we present our work Benbarka & Höfer *et al.* (2022), where we analyze the connection between the Fourier embedding and SIRENs and show that a Fourier-mapped perceptron is structurally like one hidden layer SIREN. Furthermore, we identify the relationship between the previously proposed Fourier mapping and the general d-dimensional Fourier series, leading to an integer lattice mapping. Moreover, we modify a progressive training strategy to work on arbitrary Fourier mappings and show that it improves the generalization of the interpolation task. Lastly, we compare the different mappings on the image regression and novel view synthesis tasks. We confirm the previous finding that the main contributor to the mapping performance is the size of the embedding and the standard deviation of its elements.

In the second part, we introduce our work Höfer and Zell (2022), where we propose an iterative algorithm that is able to gradually adjust a poorly chosen Fourier embedding in a way that it reaches an optimal parameterization after a few iterations. Given any parameters for a Fourier embedding, the method first finds unimportant elements via a pruning technique and then replaces them with an element adjusted in a way to optimize the overall standard deviation of the embedding.

3.1 Introduction

Real-world signals, such as images or 3D shapes, are usually represented in a discrete manner. Traditionally we represent images as a discrete set of pixels and 3D shapes as voxel grids or meshes. However, the discrete representations have a disadvantage: They are coupled to the spatial resolution, for example, it is not possible to scale a 128x128 image up to a 256x256 image, as the given information in the 128x128 image is not

enough to accurately fill in the missing pixel values. The amount of information we have about the image signal is limited by the space of the 128x128 grid.

If we instead have a continuous function f that accurately represents the image signal, that is, if we give f a pixel coordinate as input, f will output the correct RGB value for that pixel. This means that we can sample pixel grids with any resolution from f! This is also true for other signals, such as the occupancy at a pixel location in a 3D grid for the task of shape representation.

While the usage of such a function f is rather easy, it is not possible to simply write such a function down, as they typically are too complex (Sitzmann *et al.* (2019)). However, it is possible to approximate those functions with neural networks. Hence, we introduce implicit neural representations (INRs). INRs are a novel method for parameterizing signals of different types. The traditional representation of signals is discrete (e.g., a discrete set of pixels defines an image, and voxel grids or meshes define 3D shapes). The basic idea of INRs is to replace this representation with a continuous one. Specifically, one attempts to define a function from the input domain (e.g. pixel coordinates) to the respective output (e.g. the specific RGB value at the pixel location). As it is in non-trivial cases impossible to write down a mathematical formula that parameterizes these functions, INRs try to find approximations of these functions by using neural networks.

This way of representing signals has multiple advantages: The representation is not coupled to the spatial resolution anymore: having a high-resolution image traditionally increases the memory consumption, which is not the case for INRs, as they inherently have an "infinite resolution", this is especially useful in 3D and higher dimensions. Furthermore, their differentiability makes them suited for gradient-based optimization. Using INRs to represent images (Henzler *et al.* (2020); Stanley (2007)), volume density (Mildenhall *et al.* (2020)), and occupancy (Mescheder *et al.* (2019)) improves performance in multiple tasks, e.g. shape representation (Chen and Zhang (2019); Deng *et al.* (2020a); Genova *et al.* (2019, 2020); Jiang *et al.* (2020); Michalkiewicz *et al.* (2019); Park *et al.* (2019a)), texture synthesis (Henzler *et al.* (2020); Oechsle *et al.* (2019)), deriving shapes from images (Liu *et al.* (2020, 2019)), and novel view synthesis (Mildenhall *et al.* (2020); Niemeyer *et al.* (2020); Sitzmann *et al.* (2019)).

Simply building a neural network as the bridge between the input and output domain will result in poor performance on the high-frequency details of a signal. Several works, such as Mildenhall *et al.* (2020); Tancik *et al.* (2020); Sitzmann *et al.* (2020a), suggest the usage of an initial Fourier embedding to the input.

In Tancik *et al.* (2020), the authors explored the general Fourier mapping and explained why it improves the performance so dramatically using an NTK (Neural Tangent Kernel) framework. They could show that an initial Fourier embedding influences the NTK to become shift-invariant. Moreover, changing the Fourier parametrization allows tuning of the NTK spectrum and controls the complexity of details that the representation inherits. Finally, their experiments show that a Gaussian sampling with an appropriate standard deviation of the Fourier coefficients performs better than other mappings, e.g., standard positional encoding.

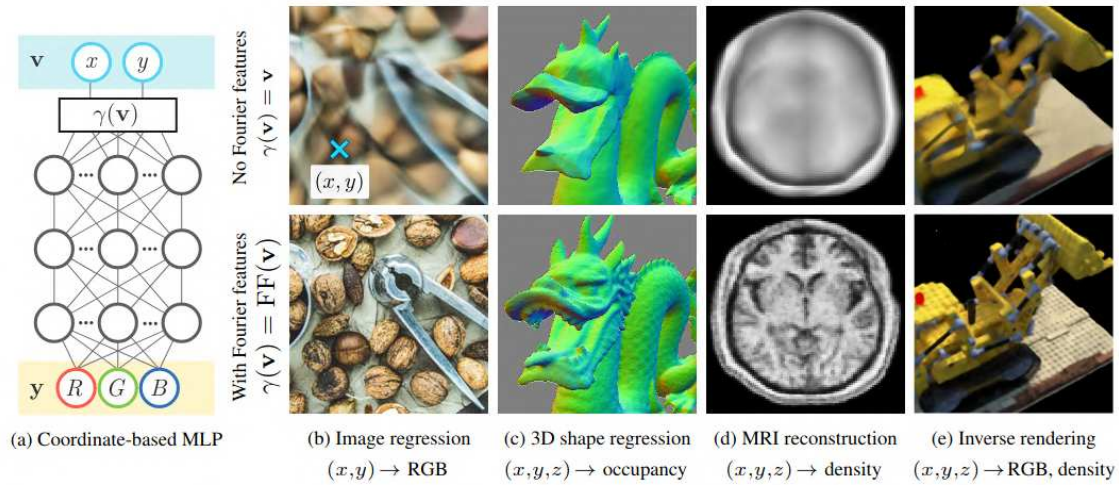


Figure 3.1: (a) Architecture of an INR designed by an example MLP for the image regression task with an initial Fourier embedding $\gamma(\cdot)$. (b)-(e) demonstrate the effect of applying an initial Fourier embedding to the input coordinates on different tasks, ranging from 3D shape regression and MRI reconstruction to inverse rendering. Superior representations are obtained in the bottom row, where a Fourier mapping was applied, as the embedding enables the representation of high-frequencies, Tancik *et al.* (2020).

To sum it up, we list the most common Fourier embeddings, given by γ, on an input signal $v \in \mathbb{R}^n$ (e.g., $v = (x, y) \in \mathbb{R}^2$ if it represents a 2D pixel location):

- The basic encoding is defined as:
  $\gamma(v) = [\cos(\pi 2v), \sin(\pi 2v)]$
  It simply wraps input coordinates around the circle.

- The positional encoding is defined as:
  $\gamma(v) = [\ldots, \cos(\pi 2^{\frac{j}{m}} v), \sin(\pi 2^{\frac{j}{m}} v), \ldots]$
  for $j = 0, \ldots, m-1$ where $m \in \mathbb{N}$, using a log-linear spacing for each dimension (Mildenhall *et al.* (2020)).

- The Gaussian embedding is defined as:
  $\gamma(v) = [\cos(2\pi \mathbf{B}v), \sin(2\pi \mathbf{B}v)]$, where $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from a normal distribution $N(0, \sigma^2)$, while σ is the hyperparameter to be optimized (Tancik *et al.* (2020)).

While these Fourier embeddings typically assume an MLP with a standard ReLU activation function, the work of Sitzmann *et al.* (2020b), that was done around the same time, proposes to make use of periodic activation functions instead of the initial Fourier

embedding. More precisely, they propose SIREN, a simple neural network architecture for INRs that makes use of the sine as its periodic activation function:

$$\phi(\mathbf{x}) = \mathbf{W}_n(\phi_{n-1} \circ \phi_{n-2} \circ \cdots \circ \phi_0)(\mathbf{x}) + \mathbf{b}_n, \quad \text{where } x_i \mapsto \phi_i(\mathbf{W}_i\mathbf{x}_i + \mathbf{b}_i) = \sin(\mathbf{W}\mathbf{x}_i + \mathbf{b}_i).$$

Here, the i^{th} layer of the network is given by $\phi_i : \mathbb{R}^{M_i} \mapsto \mathbb{R}^{N_i}$. The affine transformation is defined by the weight matrix $\mathbf{W}_i \in \mathbb{R}^{N_i \times M_i}$ and the bias $\mathbf{b}_i \in \mathbb{R}^N_i$ that are applied to the input $\mathbf{x}_i \in \mathbb{R}^{M_i}$ followed by the sine that is given as the activation function. Later we will see, that the Fourier embedding and the SIREN network are closely related, i.e., a SIREN layer represents a learnable Fourier embedding.

3.2 Related Work

The works by Park *et al.* (2019a),Chen (2019),Mescheder *et al.* (2019) first demonstrated that INRs do outperform grid-, mesh-, and point-based approaches in parameterizing geometry and allowing for learning priors over shapes. This inspired the community to develop further use cases of INRs, leading to the state of the art in 3D computer vision. Atzmon and Lipman (2020) show how we learn SDFs from raw data (i.e., without ground truth distance values). Concurrently, the works of Jiang *et al.* (2020); Peng *et al.* (2020); Chabra *et al.* (2020) proposed hybrid voxel grid/implicit representations to fit large scaled 3D scenes. Sitzmann *et al.* (2020a) showed how to parameterize 3D scenes that are room-scaled with a single implicit neural representation by leveraging sinusoidal activation functions. Interesting results are achieved in various fields, starting from 2D supervision only. Learning implicit representations of 3D shape and geometry with being given only 2D images with a differentiable ray marcher was achieved by Sitzmann *et al.* (2019). A famous work of Mildenhall *et al.* (2020) also falls in this category. They proposed the positional embedding for the Fourier embedding, as well as volumetric rendering and ray-direction conditioning for qualitative scene reconstructions, also known as Neural Radiance Fields (NeRF). A large number of follow-up publications is done on top of this work, for example, Pixel-NERF, proposed by Yu *et al.* (2021) where they proposed to condition a NeRF on local features on camera rays, which reduced the number of needed 2D images for the scene reconstruction significantly. INRs from 3D supervision that should be mentioned are Saito *et al.* (2019), which introduced the concept of conditioning INRs on local features extracted from context images. Photo realistic, real-time re-rendering was achieved by follow-up work. Also from 3D supervision, there is Texture Fields by Oechsle *et al.* (2019), which directly learns a network that maps 3D coordinates to their color value. Furthermore, INRs representing dynamic scenes were proposed by Niemeyer *et al.* (2019) using time-dependent INRs, which can be represented by space-time INRs.

When it comes to fine details of signals, the usage of a simple MLP without further modifications will yield a lack of accuracy. Hence some works tackle this fundamen-

tal problem by fitting high-frequency components with positional encoding and periodic nonlinearities. While Mildenhall *et al.* (2020) proposed positional encodings, Tancik *et al.* (2020) investigated Fourier encodings using tools from the NTK theory. With that, they could show that an MLP without an initial Fourier embedding is not able to learn complex signals. They solve this problem by showing that the usage of an initial Fourier embedding makes the kernel stationary. Still, this kernel has a tunable bandwidth and hence is able to greatly improve the performance on regression tasks. In their case, they could show that guided random mappings of the Fourier parameters achieve better results than if one takes simple positional encodings. Concurrently, Sitzmann *et al.* (2020b) proposed sinusoidal networks, which takes on a similar role as the Fourier embedding. In the following, we will find similarities between these works. In both solutions, the MLP's first layer is composed of a variant of Fourier neural networks (FNN). FNNs are neural networks that use either sine or cosine activations to get their features (Liu (2013)). The first FNN was built by Gallant and White (1988). An unsupervised hidden neural network with a cosine squasher activation function was proposed, and it was demonstrated if certain weights were hand-wired, it could represent a Fourier series. In 1999, Silvescu proposed a neural network that was different from standard feedforward neural networks. Their method, however, was to use a cosine activation function. Fourier neural networks are introduced in feedforward form by Liu (2013). They also proposed an initialization strategy of the frequencies for the embedding, which speeds up convergence. Our work will introduce another way to initialize the embedding, which results in a neural network that is precisely a Fourier series.

3.3 Seeing Implicit Neural Representations as Fourier Series

In this section, we present our work from Benbarka & Höfer *et al.* (2022), 'Seeing implicit neural representations as Fourier series'. This work aims to answer the following questions:

- What is the difference between SIRENs and the Fourier mapping?
- Will the performance be saturated when we continue to increase the mapping parameters?
- Is there a way to avoid over-fitting when training networks using Fourier mapping?
- Is a random Fourier mapping the optimal mapping?

By exploring the mathematical connection between Fourier mappings and SIRENs, we demonstrated that Fourier-mapped perceptrons are structurally similar to one hidden layer SIRENs. Unlike Fourier mappings, SIRENs are trainable, and they represent themselves in amplitude-phase form instead of sine-cosine form.

Our study of the functions we want to learn also revealed that their input domains are limited (e.g., height and width of an image), and their values are defined on a finite set of numbers. Thus, we can assume that they are continuous and periodic over their input bound, which allows us to represent them as a Fourier series. As an additional result, we have determined the trigonometric form of the d-dimensional Fourier series and shown that it is a single perceptron with an integer lattice mapping applied to its inputs. Fourier series coefficients make up the weights in that perceptron. In the same way that the Fourier series can theoretically represent any periodic signal, this perceptron can represent any periodic signal if it has an infinite number of frequencies. We can get the Fourier series coefficients in practice by sampling the signal at the Nyquist rate (double its bandwidth) and using the fast Fourier transform (FFT). As a result, the number of Fourier coefficients is the theoretical upper limit on the number of parameters required for the mapping.

Further, we modified the progressive training strategy of Lin *et al.* (2021), in which the lower frequencies are trained first, followed by the higher frequencies as the training progresses. The result of this study is that we avoid over-fitting problems with our *progressive training* strategy. The proposed *Integer Lattice* mapping was tested in the image regression and novel view synthesis tasks. We could confirm that the main contributor to the mapping performance is the number of parameters and the standard deviation, as was shown in Tancik *et al.* (2020). Our contribution can be summarized as follows:

- We introduce an integer Fourier mapping and prove that a perceptron with this mapping is equivalent to a Fourier series.
- We explore the mathematical connection between Fourier mappings and SIRENs and show that a Fourier-mapped perceptron is structurally like a one hidden layer SIREN.
- We show that the integer mapping forces periodicity of the network output.
- We modify the progressive training strategy of Lin *et al.* (2021) and show that it improves the generalization of the interpolation task.
- We compare the different mappings on the image regression and novel view synthesis tasks and verify the previous findings of Tancik *et al.* (2020) that the main contributor to the mapping performance is the number of elements and standard deviation.

Before we continue with the formulation of our method, we shortly present a mathematical Lemma, which we will need later on.

Mathematical Lemmas

Lemma 3.3.1 *The Fourier series expansion of a function with period* $\boldsymbol{p} = \mathbf{1}^d$ *defined by definition* (2.1) *reads as*

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{n} \in \mathbb{Z}^d} c_{\boldsymbol{n}} e^{2\pi i \boldsymbol{n} \cdot \boldsymbol{x}}, \tag{3.1}$$

where $c_{\boldsymbol{n}}$ *are the Fourier series coefficients given by formula* (2.2)

$$c_{\boldsymbol{n}} = \int_{[0,1]^d} f(\boldsymbol{x}) e^{-2\pi i \boldsymbol{n} \boldsymbol{x}} d\boldsymbol{x}, \tag{3.2}$$

Using Euler's formula and mathematical induction we will show that the formula can be written as:

$$f(\boldsymbol{x}) = \sum_{\boldsymbol{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1}} a_{\boldsymbol{n}} \cos(2\pi \boldsymbol{n} \cdot \boldsymbol{x}) + b_{\boldsymbol{n}} \sin(2\pi \boldsymbol{n} \cdot \boldsymbol{x}) \tag{3.3}$$

$$\begin{aligned} a_{\boldsymbol{0}} &= c_{\boldsymbol{0}}, \\ a_{\boldsymbol{n}} &= \begin{cases} 0 & \exists j \in \{2, \dots, d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2Re(c_{\boldsymbol{n}}) & otherwise, \end{cases} \\ b_{\boldsymbol{n}} &= \begin{cases} 0 & \exists j \in \{2, \dots, d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2Im(c_{\boldsymbol{n}}) & otherwise. \end{cases} \end{aligned} \tag{3.4}$$

Proof: We will prove the validity of equation (3.3) in 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. First note that for real-valued functions, it holds that $c_{\mathbf{n}} = c^*_{-\mathbf{n}}$ where $c^*_{\mathbf{n}}$ is the conjugate of $c_{\mathbf{n}}$. Furthermore, we use Euler's formula that says $e^{ix} = \cos(x) + i\sin(x)$.

$\underline{d = 1}$:

$$\begin{aligned}
f(x) &= \sum_{n\in\mathbb{Z}^1} c_n e^{2\pi i n\cdot x} \\
&= \sum_{n\in\mathbb{N}} c_n e^{2\pi i n\cdot x} + \sum_{n\in\mathbb{N}} c_{-n} e^{-2\pi i n\cdot x} + c_0 \\
&\overset{c_n^*=c_{-n}}{=} \sum_{n\in\mathbb{N}} \Big(\mathrm{Re}(c_n) + i\mathrm{Im}(c_n)\Big)\Big(\cos(2\pi nx) + i\sin(2\pi nx)\Big) \\
&\quad + \sum_{n\in\mathbb{N}} \Big(\mathrm{Re}(c_n) - i\mathrm{Im}(c_n)\Big)\Big(\cos(2\pi nx) - i\sin(2\pi nx)\Big) + c_0 \\
&= \sum_{n\in\mathbb{N}} 2\mathrm{Re}(c_n)\cos(2\pi nx) - 2\mathrm{Im}(c_n)\sin(2\pi nx) + c_0 \\
&= \sum_{n\in\mathbb{N}_0} a_n\cos(2\pi nx) + b_n\sin(2\pi nx),
\end{aligned} \tag{3.5}$$

where

$$a_0 = c_0,\ a_n = 2\mathrm{Re}(c_n),\ b_n = -2\mathrm{Im}(c_n). \tag{3.6}$$

Assumption of the induction:
We will assume that the equation (3.3) holds for $d-1$, where $d \geq 2$.

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

$$\begin{aligned}
&\sum_{\mathbf{n}=(n_1,\ldots,n_d)\in\mathbb{Z}^d} c_{\mathbf{n}} e^{2\pi i \mathbf{n}\cdot\mathbf{x}} \\
&\overset{(*)}{=} \sum_{n_1\in\mathbb{N}} \sum_{(n_2,\ldots,n_d)\in\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i \mathbf{n}\cdot\mathbf{x}} + \sum_{n_1\in\mathbb{N}} \sum_{(n_2,\ldots,n_d)\in\mathbb{Z}^{d-1}} c_{-\mathbf{n}} e^{-2\pi i \mathbf{n}\cdot\mathbf{x}} \\
&\quad + \sum_{n_1=0}^{0} \sum_{(n_2,\ldots,n_d)\in\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i \mathbf{n}\cdot\mathbf{x}} \\
&\overset{c_{\mathbf{n}}^*=c_{-\mathbf{n}}}{=} \sum_{\mathbf{n}\in\mathbb{N}\times\mathbb{Z}^{d-1}} 2\mathrm{Re}(c_{\mathbf{n}})\cos(2\pi\mathbf{n}\cdot\mathbf{x}) - 2\mathrm{Im}(c_{\mathbf{n}})\sin(2\pi\mathbf{n}\cdot\mathbf{x}) \\
&\quad + \sum_{\mathbf{n}\in\{0\}\times\mathbb{Z}^{d-1}} c_{\mathbf{n}} e^{2\pi i \mathbf{n}\cdot\mathbf{x}} \\
&\overset{\text{Ind. asm.}}{=} \sum_{\mathbf{n}\in\mathbb{N}\times\mathbb{Z}^{d-1}} 2\mathrm{Re}(c_{\mathbf{n}})\cos(2\pi\mathbf{n}\cdot\mathbf{x}) - 2\mathrm{Im}(c_{\mathbf{n}})\sin(2\pi\mathbf{n}\cdot\mathbf{x}) \\
&\quad + \sum_{\mathbf{n}\in\{0\}\times\mathbb{N}_0\times\mathbb{Z}^{d-2}} a'_{\mathbf{n}}\cos(2\pi\mathbf{n}\cdot\mathbf{x}) + b'_{\mathbf{n}}\sin(2\pi\mathbf{n}\cdot\mathbf{x}),
\end{aligned} \tag{3.7}$$

where

$$
\begin{aligned}
a'_{\mathbf{0}} &= c_{\mathbf{0}}, \\
a'_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{3,\dots,d\} : n_2 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2\mathrm{Re}(c_{\mathbf{n}}) & \text{otherwise,} \end{cases} \\
b'_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{3,\dots,d\} : n_2 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2\mathrm{Im}(c_{\mathbf{n}}) & \text{otherwise.} \end{cases}
\end{aligned} \tag{3.8}
$$

Combining these two summands, we get

$$
\sum_{\mathbf{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1}} a_{\mathbf{n}} \cos(2\pi \mathbf{n} \cdot \mathbf{x}) + b_{\mathbf{n}} \sin(2\pi \mathbf{n} \cdot \mathbf{x}), \tag{3.9}
$$

where

$$
\begin{aligned}
a_{\mathbf{0}} &= c_{\mathbf{0}}, \\
a_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2\mathrm{Re}(c_{\mathbf{n}}) & \text{otherwise,} \end{cases} \\
b_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2\mathrm{Im}(c_{\mathbf{n}}) & \text{otherwise.} \end{cases}
\end{aligned} \tag{3.10}
$$

□

3.3.1 Method

Integer Lattice Mapping

This section explains how a perceptron with an integer lattice Fourier mapping translates to a Fourier series. The Fourier-mapped perceptron equation is presented first, followed by the Fourier series' general equation. The perceptron is the fundamental building block of any neural network, and it is defined as

$$
\mathbf{y}(\mathbf{x}, \mathbf{W}', \mathbf{b}) = g(\mathbf{W}' \cdot \mathbf{x} + \mathbf{b}). \tag{3.11}
$$

Here $\mathbf{y} \in \mathbb{R}^{d_{out}}$ is the perceptron's output, $g(\cdot)$ is the (usually non-linear) activation function, $\mathbf{x} \in \mathbb{R}^{d_{in}}$ 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 obtain

$$
\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W} \cdot \gamma(\mathbf{x}) + \mathbf{b}, \tag{3.12}
$$

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

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi\mathbf{B}\cdot\mathbf{x}) \\ \sin(2\pi\mathbf{B}\cdot\mathbf{x}) \end{pmatrix}. \tag{3.13}$$

$\mathbf{W} \in \mathbb{R}^{d_{out}\times 2m}$, $\mathbf{B} \in \mathbb{R}^{m\times d_{in}}$ is the Fourier mapping matrix, and m is the number of frequencies. Equation 3.12 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. The functions we want to learn in applications that use coordinate-based MLPs are not periodic. However, their inputs are naturally bounded (e.g., the height and width of an image). Accordingly, it does not 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

$$f(\mathbf{x} + \mathbf{n} \circ \mathbf{p}) = f(\mathbf{x}) \quad \forall \mathbf{n} \in \mathbb{Z}^d, \tag{3.14}$$

where $\circ$ is the Hadamard product. As it is plausible to normalize the inputs with respect to their bounds, we assume that each variable's period is 1. The Fourier series expansion of function (3.14) with $\mathbf{p} = \mathbf{1}^d$ is defined in (2.1) to be

$$f(\mathbf{x}) = \sum_{\mathbf{n}\in\mathbb{Z}^d} c_{\mathbf{n}} e^{2\pi i \mathbf{n}\cdot\mathbf{x}}, \tag{3.15}$$

where $c_{\mathbf{n}}$ are the Fourier series coefficients, given by (2.2)

$$c_{\mathbf{n}} = \int_{[0,1]^d} f(\mathbf{x}) e^{-2\pi i \mathbf{n}\mathbf{x}} d\mathbf{x}. \tag{3.16}$$

Using Euler's formula and mathematical induction Lemma 3.3.1 tells us that equation (3.15) can be written as

$$f(\mathbf{x}) = \sum_{\mathbf{n}\in\mathbb{N}_0\times\mathbb{Z}^{d-1}} a_{\mathbf{n}} \cos(2\pi\mathbf{n}\cdot\mathbf{x}) + b_{\mathbf{n}} \sin(2\pi\mathbf{n}\cdot\mathbf{x}), \tag{3.17}$$

$$
\begin{aligned}
a_{\mathbf{0}} &= c_{\mathbf{0}}, \\
a_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ 2\mathrm{Re}(c_{\mathbf{n}}) & \text{otherwise,} \end{cases} \\
b_{\mathbf{n}} &= \begin{cases} 0 & \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0 \\ -2\mathrm{Im}(c_{\mathbf{n}}) & \text{otherwise.} \end{cases}
\end{aligned} \tag{3.18}
$$

Writing equation (3.17) in vector form, we receive

$$
f(\mathbf{x}) = (a_{\mathbf{B}}, b_{\mathbf{B}}) \cdot \begin{pmatrix} cos(2\pi \mathbf{B} \cdot \mathbf{x}) \\ sin(2\pi \mathbf{B} \cdot \mathbf{x}) \end{pmatrix}, \tag{3.19}
$$

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 3.12 and 3.19, 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 $\mathbf{n}$. For practicality, we limit $\mathbf{B}$ to

$$
\mathbf{B} = \{0,\dots,N\} \times \{-N,\dots,N\}^{d-1} \setminus \mathbf{H}, \tag{3.20}
$$

where N will be called the frequency of the mapping, the set $\mathbf{H}$ is defined as $\mathbf{H} = \{\mathbf{n} \in \mathbb{N}_0 \times \mathbb{Z}^{d-1} | \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}$. Then the perceptron represents a Fourier series. Hence, we calculate the dimension m of all possible permutations

$$
m = (N+1)(2N+1)^{d-1} - \sum_{l=0}^{d-2} N(2N+1)^l. \tag{3.21}
$$

We will prove this with mathematical induction. 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

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

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, that

$$
|\{\mathbf{n} \in [\![N]\!] \times [\![-N,N]\!]^{d-1} : \exists j \in \{2,\dots,d\} : n_1 = \cdots = n_{j-1} = 0 \wedge n_j < 0\}| \overset{!}{=} \sum_{l=0}^{d-2} N(2N+1)^l. \tag{3.22}
$$

We will do this proof with mathematical induction. As $d = 1$ is trivial 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 (3.22) holds for some d, where $d \geq 2$.

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

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

The Fourier series coefficients can be found in practice by sampling a function uniformly at a frequency higher than the Nyquist frequency and performing a Fast Fourier Transform (FFT) on the sampled signal. An FFT coefficient is the Fourier series coefficient multiplied by the number of sampled points. We should, in theory, achieve our training target after iteration zero if we initialize the weights with the Fourier series coefficients.

SIRENs and Fourier Mapping Comparison

In this section we show that a Fourier mapped perceptron is structurally like a SIREN with one hidden layer, as it is displayed in figure 3.2. If we evaluate $\mathbf{W} \cdot \gamma(\mathbf{x})$ in equation (3.12), using (3.13) and combine the sine and cosine terms, we get:

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \mathbf{W} \cdot sin(2\pi \mathbf{C} \cdot \mathbf{x} + \phi) + \mathbf{b}, \tag{3.23}$$

where $\phi := (\pi/2, \ldots, \pi/2, 0, \ldots, 0)^T \in \mathbb{R}^{2m}$ and $\mathbf{C} := (\mathbf{B}, \mathbf{B})^T$. 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

Figure 3.2: Visualization that the initial SIREN layer with a sinusoidal activation function is the same as a Fourier mapped perceptron with a learnable Fourier embedding $\gamma^*(\mathbf{x})$.

case. This finding closes the bridge between Fourier frequency mappings and sinusoidal activation functions, which have recently attracted a lot of attention.
The proof of this formula is a straightforward calculation. By combining equation (3.12) and (3.13) 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

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

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

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

Progressive Training

Lin *et al.* (2021) introduced a training strategy for coarse-to-fine registration for NeRFs which they called BARF. Their approach is to mask out the positional encoding's high-

frequency activations at the start and gradually allow them as training progresses. This strategy was shown to improve camera registration only on positional encodings. In our work, we will demonstrate how to use this strategy on any Fourier mapping and show that it improves interpolation generalization. We weight the frequencies of γ as follows:

$$\gamma^{\alpha}(\mathbf{x}) := \begin{pmatrix} w^{\alpha}_{\mathbf{B}} \\ w^{\alpha}_{\mathbf{B}} \end{pmatrix} \circ \gamma(\mathbf{x}) \tag{3.24}$$

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:

$$w^{\alpha}_{\mathbf{B}} := w_{\alpha} \begin{pmatrix} ||B_1||_2 \\ \vdots \\ ||B_m||_2 \end{pmatrix}. \tag{3.25}$$

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

$$w_{\alpha}(z) = \begin{cases} 0 & \text{if } \alpha - z < 0 \\ \frac{1-\cos((\alpha-z)\pi)}{2} & \text{if } 0 \leq \alpha - z \leq 1 \\ 1 & \text{if } \alpha - z > 1 \end{cases} \tag{3.26}$$

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 high-frequency components are trained, and the network can focus on the left details. This strategy should reduce the effect of overfitting. It was introduced by Tancik *et al.* (2020) when using mappings with large standard deviations.

Perceptron Pruning

The standard way of using equation (3.20) 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 perceptron 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.

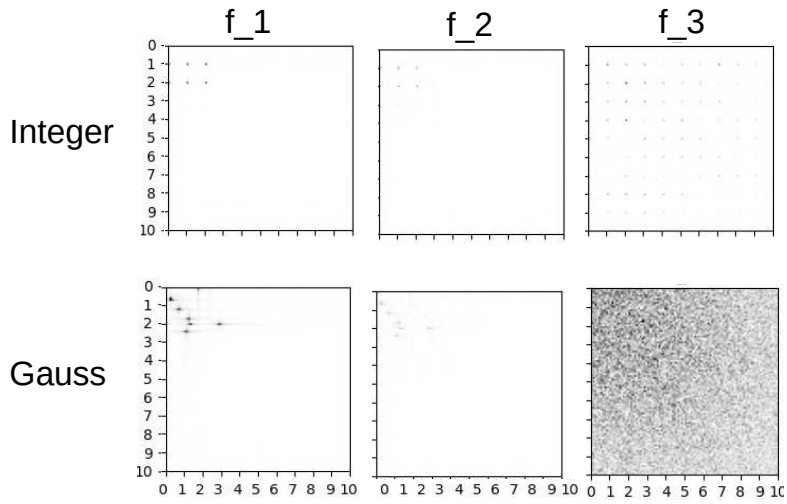


Figure 3.3: The effect of the common activation functions on the spectrums of functions with integer and non-integer frequencies. The first row are visualizations with an integer embedding and the second row shows the spectrum for an initial Gaussian embedding. The tested functions are: f1 is the identity function, f2 is the ReLU activation and f3 is the sine.

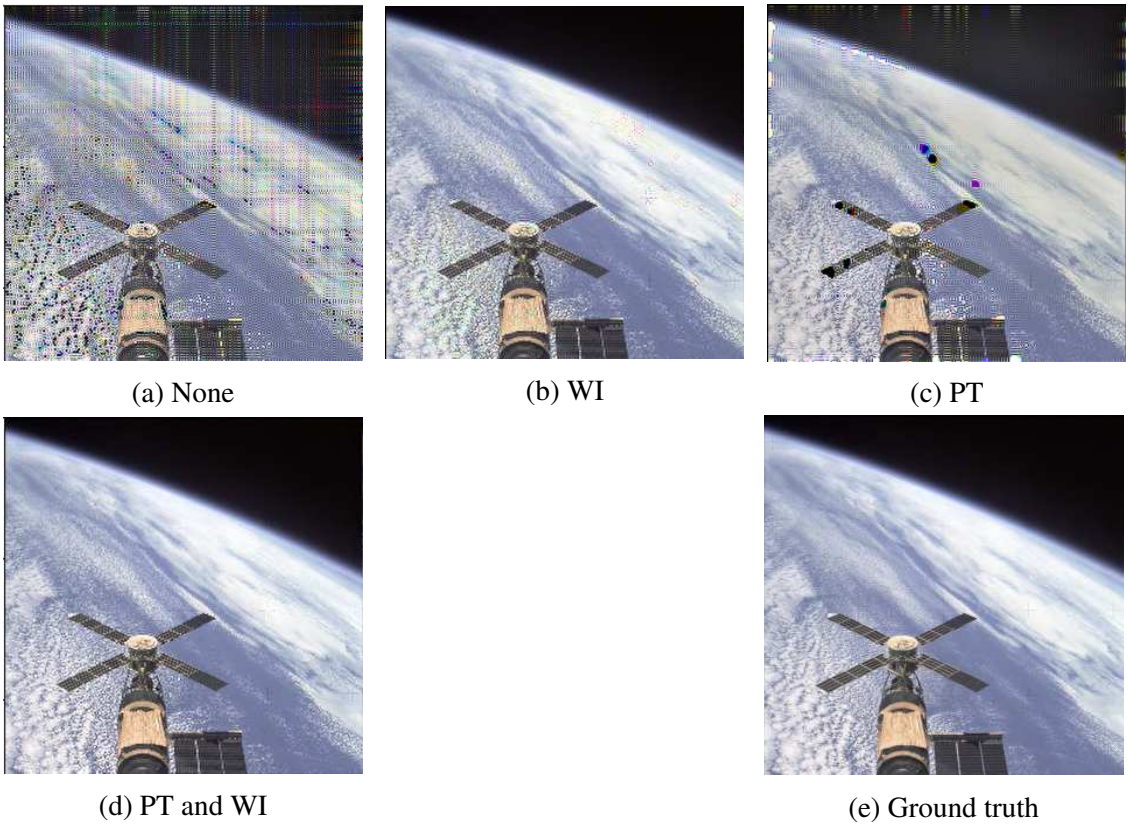
Integer Lattice Mapping Applied to MLPs

Although we showed in section 3.3.1 that we can 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 performance. However, it remains to be seen 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.

Periodicity in MLPs

The reader is reminded that a periodic function has integer frequencies. Our assumption of a periodic signal indicates that it will have integer frequencies. Also, the activation functions we are using only introduce integer frequencies when applied to a periodic function, as we will show for the 1D case. 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



(a) None (b) WI (c) PT

(d) PT and WI (e) Ground truth

Figure 3.4: A visualization of the outputs of Fourier mapped perceptrons of $N = 128$. PT stands for progressive training and WI stands for weight initialization.

is more compact and approachable. We claim that the network output is compelled to be periodic when an integer mapping is applied to the input. This results from the fact that a periodic signal only has integer frequencies and that the frequencies introduced by the activations are integers. We will first examine the frequencies in the 1D example to support this assertion, and we will then use 2D tests to support our conclusions. We now explore the impact of putting an activation function on top of a sinus representation, as an initial Fourier mapping entails using a sinus function on the mapped input. Frequencies that are multiples of the input frequencies are produced when a ReLU or Sine is applied to a mapped input.

$$ReLU(\sin(x)) = \frac{1}{\pi} + \frac{\sin(x)}{2} + \sum_{\substack{n=2k \\ k\in\mathbb{N}}} \frac{2}{\pi(1-n^2)}\cos(nx), \tag{3.27}$$

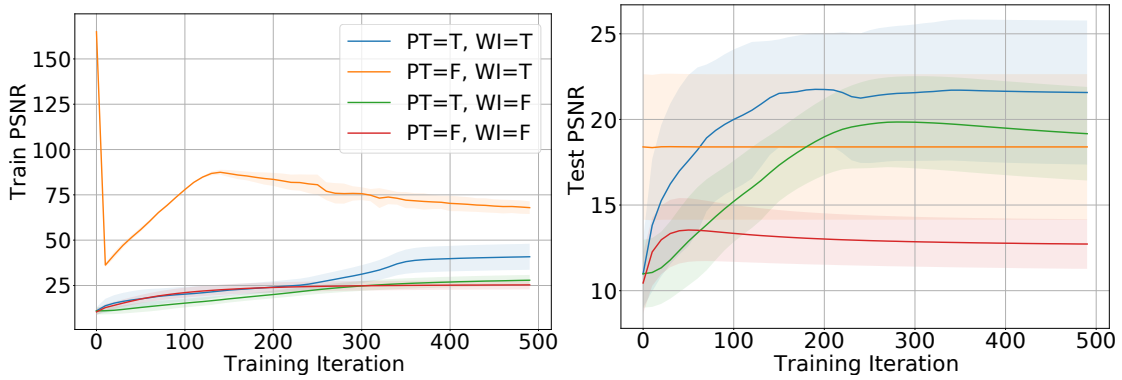


Figure 3.5: The training progress of Fourier mapped perceptrons with $N = 128$. The left and the right figures report the train and test PSNR, respectively. Weight initialization without PT yields a PSNR of 160 which one can consider as the ground truth proving that the perceptron is a Fourier series. Note that the y-axis limits are different in both plots.

and if we apply a sine to a sine, we obtain

$$\sin(A \cdot \sin(x)) = 2 \sum_{n=0}^{\infty} J_{2n+1}(A) \sin((2n+1)x), \tag{3.28}$$

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 the question of whether it does generalize to higher dimensional signals.

To do so, we define the Fourier matrix $\mathbf{B}$ in two different ways. First, we generate $\mathbf{B}_N$ limited by $N = 2$, responsible for the integer mapping and the Gauss mapping, we 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. 3.3 in the range of $(0, 10)^2$.

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

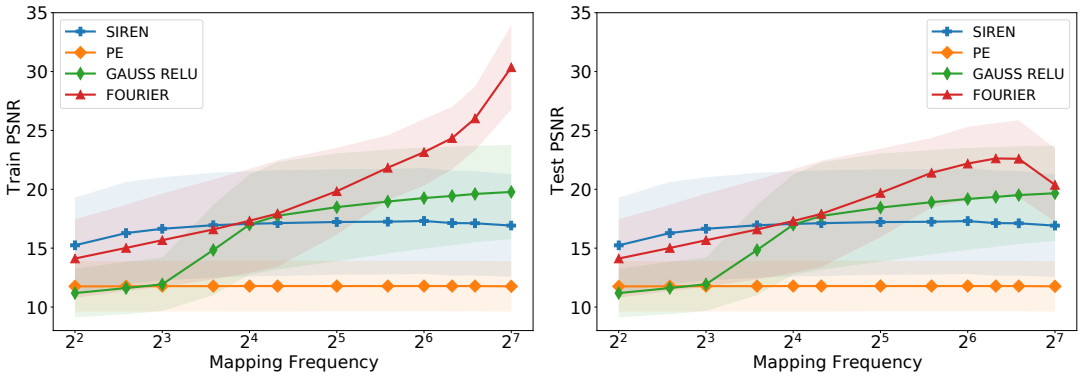
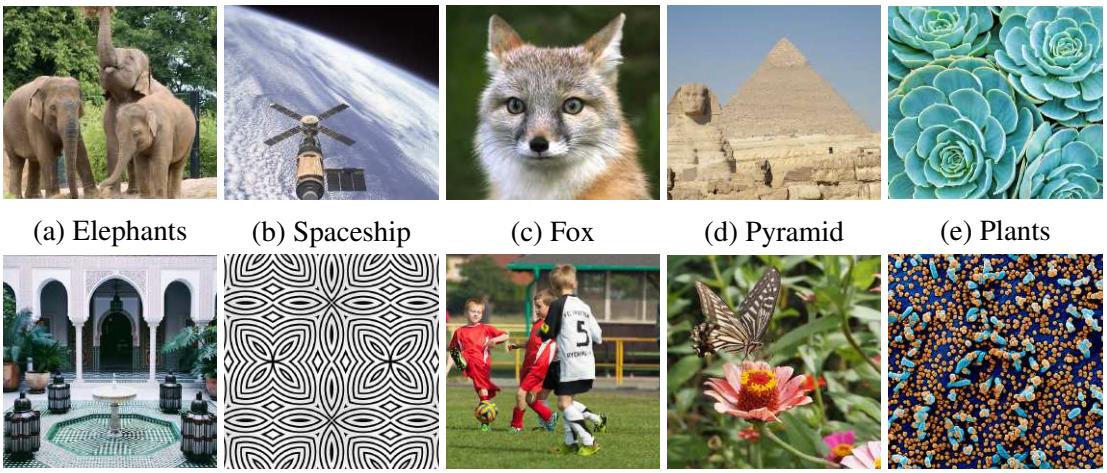


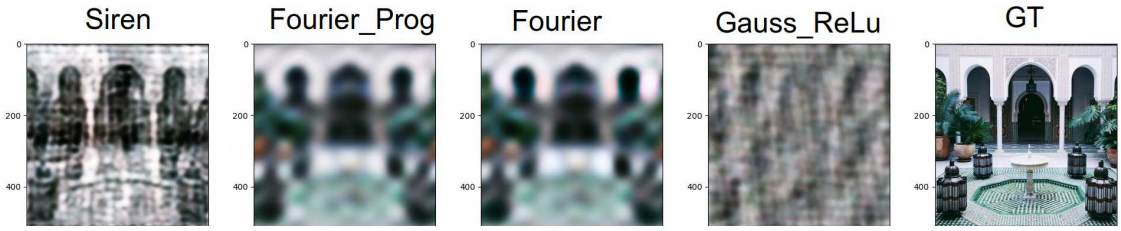
Figure 3.6: Perceptron experiments with different values for the mapping frequency N. We report the train PSNR on the left and the test PSNR on the right. For high values of N our integer mapping outperforms all competing mappings.



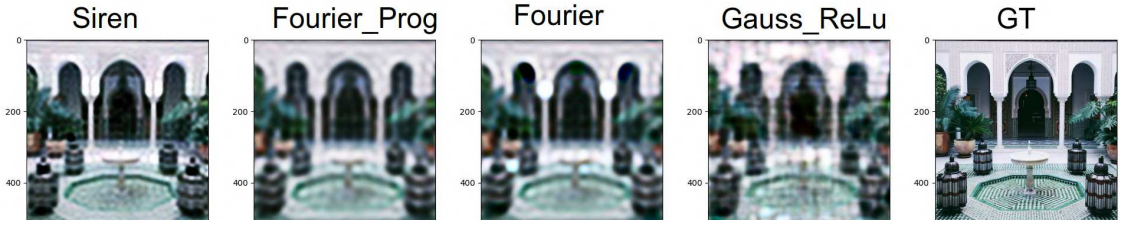
(a) Elephants (b) Spaceship (c) Fox (d) Pyramid (e) Plants

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

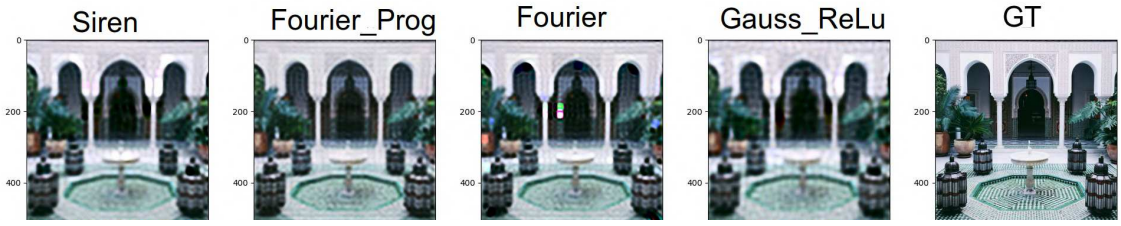
Figure 3.7: The images used in the image regression experiments.



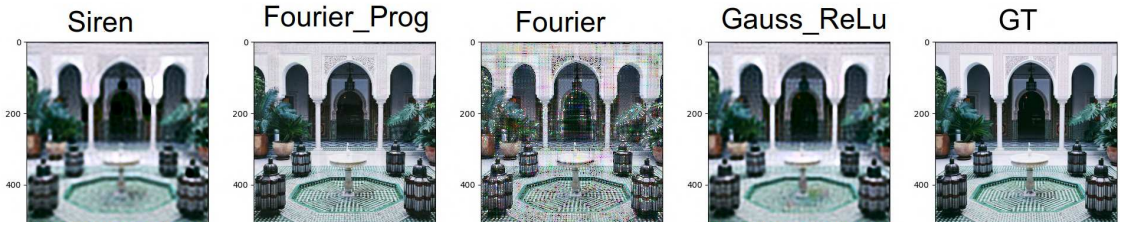
(a) Network predictions using $N = 8$



(b) Network predictions using $N = 16$



(c) Network predictions using $N = 32$



(d) Network predictions using $N = 128$

Figure 3.8: The visualization of the Fourier mapped perceptrons and the one layer Siren with different numbers of parameters. Fourier_Prog is trained with the progressive training scheme.

3.3.2 Experiments

Weight Initialization and Progressive Training

In this section, we want to confirm 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 neural network memorize an image by predicting the color at each pixel location. We use ten images with a resolution of 512×512, figure 3.7 shows the images used for the image regression task. We report the mean peak signal-to-noise ratio (PSNR). We divide the images into train and test sets, where we use every second pixel for training and take the complete image for testing. We utilize three Fourier-mapped perceptrons with $N = 128$ (Nyquist frequency), one for each image channel. We normalize the input ($\mathbf{x}$) to be between [0,1] in both width (x) and height (y) dimensions.

In this experiment, we made an ablation: With and without weight initialization using the normalized FFT coefficients of the images' training pixels, with and without the progressive training scheme explained in section 3.3.1. For progressive training, α was linearly increased from 0 to its maximum value at 75% of training iterations. During 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 try different schedules. Figure 3.4 shows a visualization of one of the images, and Figure 3.5 shows the training progress, where the solid line is the mean PSNR and the shaded area shows the standard deviation.

Figure 3.5 shows 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*. In case 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 3.5 that whenever we use progressive training, it always shows a higher test PSNR, which certifies that *progressive training helps with generalization*. Lastly, when we did not employ both PT and WI, the perceptron overfits to the training pixels, and this can be seen quantitatively with a very low test PSNR (red line in Figure 3.5) and qualitatively with grid-like artefacts (in Figure 3.4a)).

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 3.3.2, where we used progressive training and did not use weight initialization.

In the integer mapping, we increased the value of N from 4 until half the training image dimension (Nyquist frequency) and calculated all possible permutations $\mathbf{B}_N$, as discussed in section 3.3.1. For the Gaussian mapping, we sample $m = |\mathbf{B}_N|$ parameters

Activ.	Mapping	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
	Integer	15.68	**22.31**	22.41	20.94	**17.33**	27.66	27.06	**27.33**	**19.84**	33.78	26.98	23.60
	Pruned	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	Pos. Enc.	11.78	16.61	17.37	17.77	11.78	16.87	17.79	17.95	11.78	17.05	18.15	18.15
	Gauss 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
	Gauss 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**
	Integer	15.68	20.51	20.65	20.62	**17.33**	24.42	24.09	24.49	**19.84**	31.57	32.14	32.79
	Pruned	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 3.1: The mean train 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.	Mapping	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
	Integer	15.68	**21.75**	21.53	20.06	**17.31**	**23.48**	**22.67**	22.28	**19.70**	16.85	17.89	16.36
	Pruned	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	Pos. Enc.	11.78	16.60	17.33	17.70	11.78	16.85	17.73	17.87	11.79	17.02	18.06	18.02
	Gauss 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**
	Gauss 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
	Integer	15.68	20.27	20.35	20.23	**17.31**	22.93	22.65	**22.50**	**19.70**	**24.36**	**24.02**	**23.73**
	Pruned	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 3.2: The mean test PSNR results of network type comparison experiment. For abbreviations see table 3.1.

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 the same size m. Finally, we did not include the positional encoding (PE) scheme from Mildenhall *et al.* (2020), as it was not able to regress the image in the perceptron case. Figure 3.6 shows our experiments' results on the train and test pixels, respectively. Figure 3.8 shows the networks' outputs trained on one of the images.

At low N values (Figure 3.8a), we see that the Gaussian mapped perceptrons do not work because the number of sampled frequencies is 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 3.3.1. 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.

However, at around $N = 30$, the SNR of the SIREN and the Gaussian-mapped perceptrons saturates, while the integer-mapped perceptrons' PSNR keeps rising. 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. Furthermore, the integer mapping matrix's standard deviation increases as N increases, and we see that the integer mapped perceptrons with the usual training scheme start to overfit. And these results agree with the conclusions given by Tancik *et al.* (2020). However, when we used the progressive training strategy, the perceptron avoided overfitting and achieved the best training and test PSNRs. Even the trainability of the SIREN mapping did not help in this case.

MLP Experiments

Our theory for integer mapping assumes an underlying function that is periodic. However, it needs to be clarified that we will end up with a periodic function if we go the other way, using 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 3.3.1) 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, positional encoding (PE), and pruned integer mapping (section 3.3.1). We also compared SIRENS with no mapping (extra layer), integer, or 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. For the pruned mapping, we used a $pr(N, 128)$. And 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 3.1 and 3.2 show

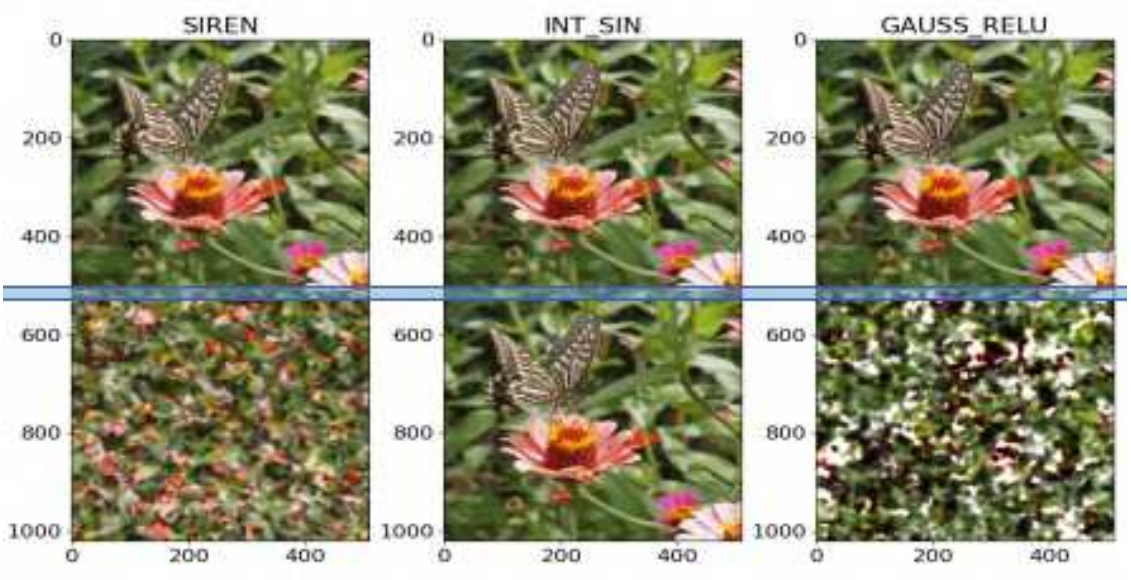


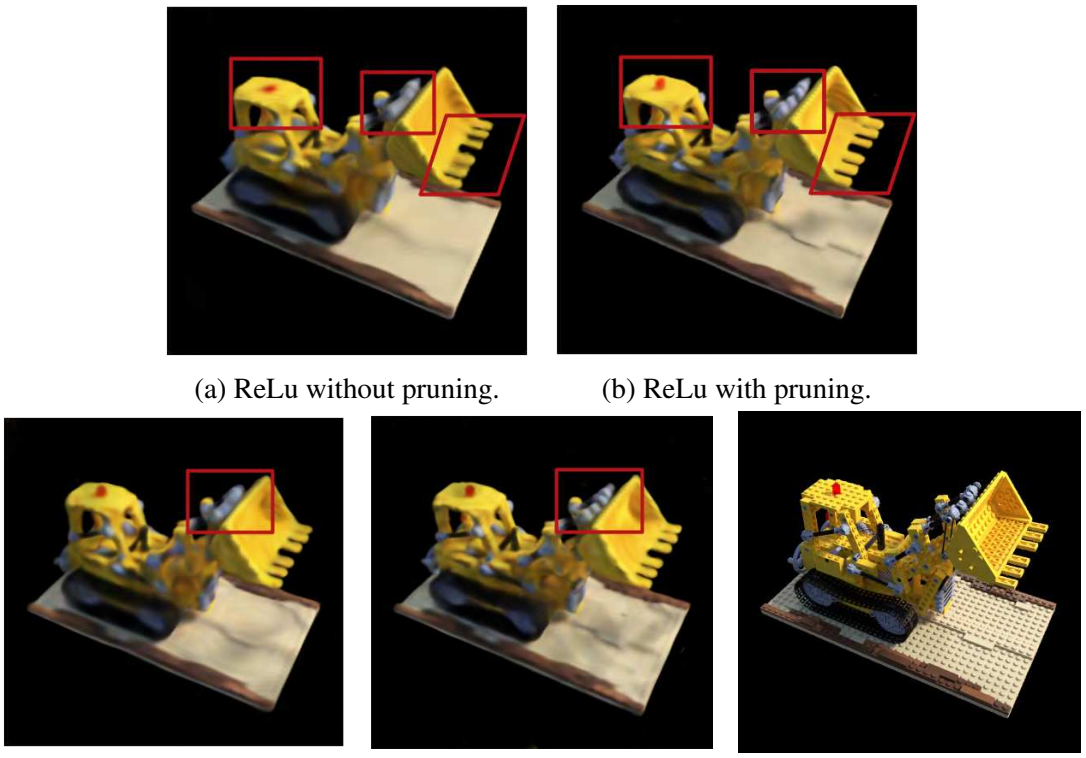
Figure 3.9: Comparison of the SIREN (Sitzmann *et al.* (2020a)), Gauss ReLU (Tancik *et al.* (2020)) and our results on the image regression task. The row above the horizontal line shows the reconstructed 512×512 image of the first period, and the row below the horizontal line shows the next period in the height and width dimensions.

the mean train and test PSNRs, respectively.

Figure 3.9 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 the SIREN and ReLU network with Gauss mapping, which proves our first hypothesis.

From the table 3.1, 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 also see that the pruned integer mapping has comparable results with the Gauss σ_{pr}, which 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



(a) ReLu without pruning. (b) ReLu with pruning.

(c) Sine without pruning. (d) Sine with pruning. (e) Ground truth

Figure 3.10: 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.

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.

Novel View Synthesis Experiments

This section analyzes whether our findings in the image regression task transfer to the novel view synthesis (NVS) task. In NVS, we are given a set of 2D images of a scene, and we are supposed to find its 3D representation. With this representation, one can render images from new viewpoints. In contrast to the 2D experiments, the inputs are (x, y, z) coordinates that are mapped to a 4-dimensional output, the RGB-values, and a volume density. For this experiment, a simplified version of the official NeRF from Mildenhall *et al.* (2020) is used, where the view dependency and hierarchical sampling are removed. Here, we experiment with the input mappings used in section 3.3.2. Unless otherwise stated, we adopt the settings from the image regression task. We set the network width to 64.

As the mapping size increases exponentially, we do our experiments with lower frequencies than in the 2D case. We used the integer mapping on four frequencies. The

Act.	Mapping	$N = 4$				N=8
		d=0	d=2	d=4	d=6	d=0
Sine	No	**20.37**	23.08	**23.55**	23.35	OM
	Integer	18.42	22.22	22.95	22.97	**19.31**
	Pruned	19.15	**23.12**	**23.58**	23.36	-
Relu	Pos. Enc.	16.30	21.48	22.64	23.51	16.40
	Gauss	18.93	22.81	**23.64**	**23.82**	19.29
	Integer	18.42	21.81	22.68	23.28	**19.31**
	Pruned	19.15	22.78	23.61	**23.89**	**-**

Table 3.3: Validation PSNR scores of Nerf experiments using a mapping of frequency 4. OM stands for out of memory.

frequencies of our mapping were limited to the maximum network size which we could fit on an NVIDIA RTX-2080Ti with 11GB memory. The pruning is given by $pr(4,8)$. We conduct our experiments on the bulldozer scene, which is commonly used for Nerf experiments. For training, we used a batch size of 128 for 50.000 epochs and a learning rate of 5×10^{-4}.

As seen in Table 3.9, 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 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. Qualitative improvements of pruning can be seen in Figure 3.10. 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, due to memory limitations, we could not test a perceptron with frequencies higher than 8, which was superior in image regression.

3.3.3 Conclusion

In this work, we identified a relationship between the Fourier mapping and the general d-dimensional Fourier series, which led to the *integer lattice mapping*. We also showed that this mapping forces periodicity of the neural network's underlying function. From experiments, we showed that one perceptron with frequencies equal to the Nyquist rate of the signal is enough to reconstruct it. Furthermore, we showed that the progressive training strategy improves the generalization of the interpolation task. Lastly, we confirmed the previous findings that the main contributor to the mapping performance is its size and the standard deviation of its elements.

3.4 Automatic Adjustment of Fourier Embeddings

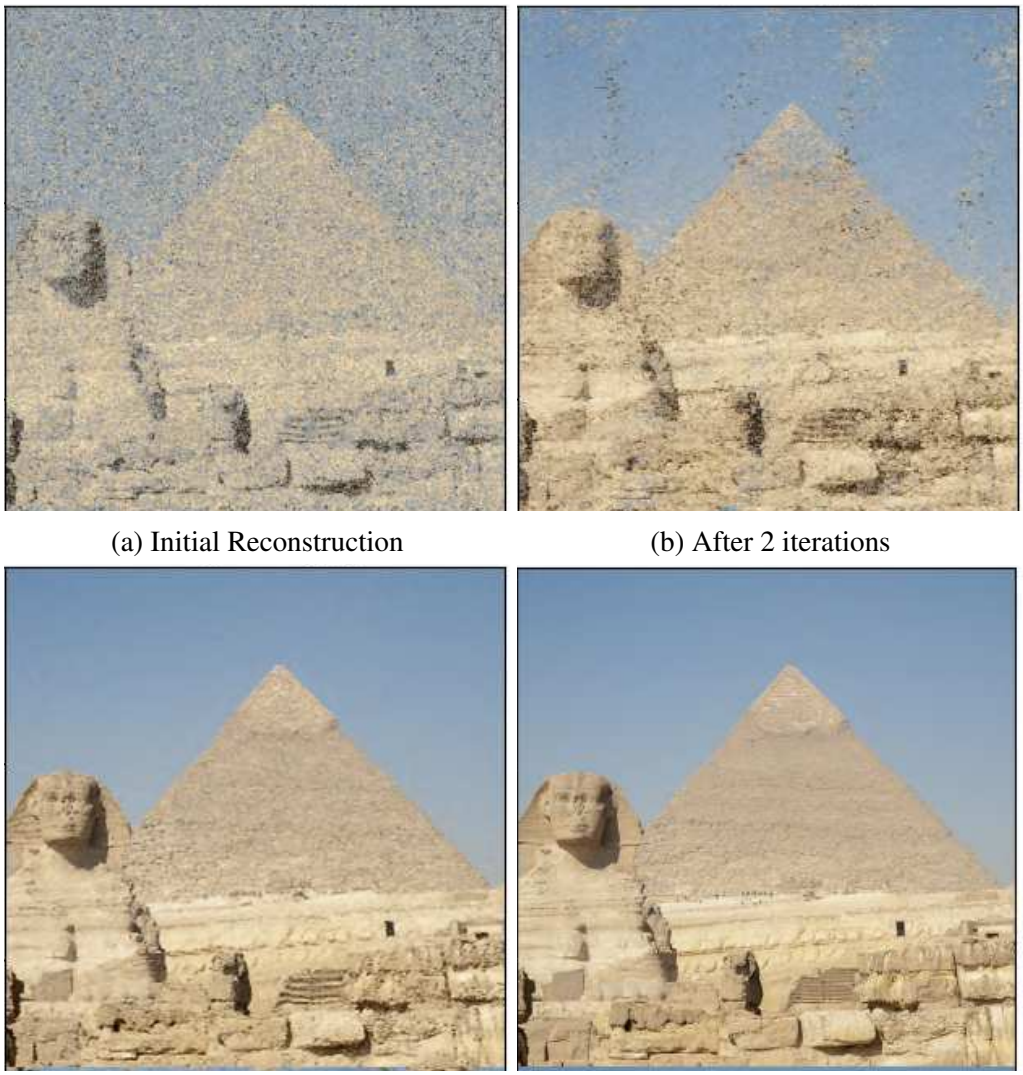
While everything points towards the usage of Fourier embeddings, it still requires expert knowledge when choosing the optimal embedding. Naively applying positional encoding can become a double-edged sword, as stated by Lin *et al.* (2021). This also applies to the Gaussian embedding, following Tancik *et al.* (2020), where it requires an initial hyperparameter search on the standard deviation used for the sampling of the parameters. The previously introduced integer mapping from Benbarka & Höfer *et al.* (2022) suffers from a high memory consumption in higher-dimensional problem domains. Hence a trade-off between the number of parameters and available resources has to be made.

In this section we want to present the work of Höfer and Zell (2022), which aims to skip the search for the correct embedding parameters by a remove-and-replace approach during training. Starting with some Fourier embedding matrix $\mathbf{B}$, we will discuss the question of which of its elements are the most influential ones for the performance. After removing the least essential elements, we replace them with their adjusted version. This adjustment procedure mainly relies on findings of Tancik *et al.* (2020) and Benbarka & Höfer *et al.* (2022), namely that the highest influence on the performance is the standard deviation of the Fourier parameters. Our contributions are as follows:

- We introduce and analyze different pruning techniques that determine the importance of a Fourier embedding parameter.
- We propose a replacement strategy for unimportant Fourier embedding parameters to increase the overall performance.
- Finally, we formulate the remove and replace techniques as an iterative method, removing the need to search for an optimal Fourier embedding parametrization.

Still, it remains unclear which parametrization to choose to achieve the best performance for the positional encoding and the Gaussian sampling. An optimal parametrization is dependent on many factors, like the task and the network architecture. In the previous section, we proposed the usage of the progressive training strategy for coarse-to-fine registration. Starting with a fixed initial Fourier parametrization, high frequencies are masked out at the beginning of training and are then gradually included. This is particularly useful for applications starting with unknown poses, which can be the case in view synthesis and localization of video sequences. While this allows adjustment of the Fourier parameters, this strategy is still dependant on the initial parametrization.

This work will focus on the Gaussian encoding using Fourier features, shown to outperform the positional encoding Tancik *et al.* (2020), as it is the most challenging one without an underlying structure. Different from the previous works, we want to allow the Fourier embedding parameters to adapt. With this, the need to search for an optimal initial parametrization is redundant as the network itself searches for its optimal parametrization through an iterative method.



(a) Initial Reconstruction (b) After 2 iterations

(c) After 5 iterations (d) Ground truth

Figure 3.11: Reconstruction results on the image regression task. An initial Fourier embedding matrix produces poor reconstructions (a). After two iterations of our adjustment method, more details are visible (b). Finally, after five iterations, our method produces high-quality results (c) that visually matches the ground truth (d).

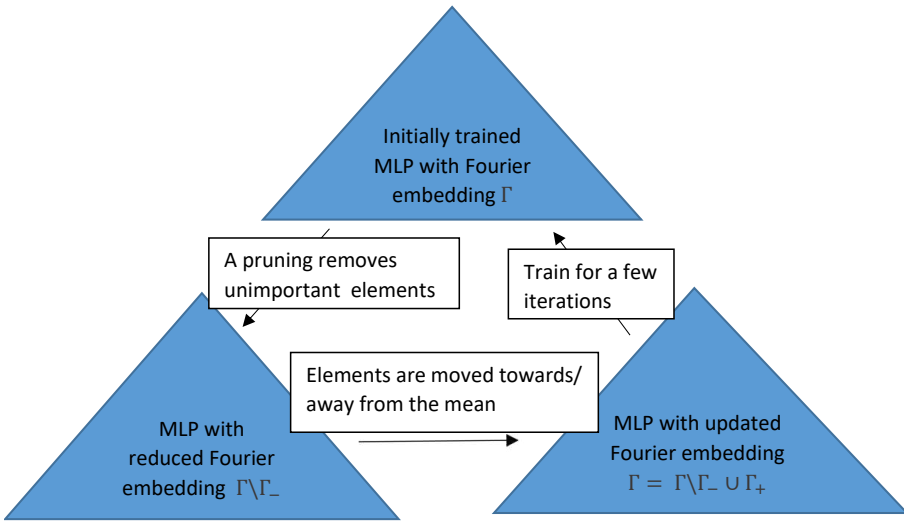


Figure 3.12: Overall Remove-and-Replace methodology. After training the network for a few iterations, we remove the Fourier embedding matrix elements via the removal strategy from section 3.4.2. To keep the size constant, we replace those elements with the replacement strategy from section 3.4.2 and continue training the MLP. This is an iterative procedure.

3.4.1 Method

The Fourier Embedding

Instead of using the pixel coordinates directly as input to the network, it is common to use an initial Fourier embedding, as it is displayed in Figure 3.1. In this work, we focus on the Gaussian Fourier embedding, motivated by Tancik *et al.* (2020). We remind the reader about its definition in the following: Let v be the input coordinate of the corresponding signal, e.g. the pixel coordinate in the image regression task. The dimension of the coordinate depends on the task. E.g. in the image regression task, it is 2-dimensional. Then, the Gaussian embedding is defined as follows:

$$\Gamma(v) = [\cos(2\pi \mathbf{B}v)^T, \sin(2\pi \mathbf{B}v)^T],$$

where $\mathbf{B} \in \mathbb{R}^{m \times d}$ and each entry is sampled from $N(0, \sigma^2)$, with σ as its hyperparameter.

3.4.2 Overall Methodology

Our general idea is to provide an iterative method that consists of a remove and replace strategy used during training that can adjust an initial parameterization of the Fourier

embedding so that the overall performance increases. To do so, we define ways to select the elements to be removed and the elements by which they are replaced.

Following Figure 3.12, the general pipeline is as follows: We start training with an arbitrary and not optimized initialization of the Fourier parameters. After N training iterations, we select k elements to be removed and replace them immediately. As this is an iterative method, one can either set a fixed number of iterations or guarantee convergence by tracking the performance between the iteration steps and stopping whenever the training/validation accuracy drops.

Which Elements Are to Be Removed?

To select the least essential elements of the embedding matrix **B**, it is crucial to have a way to measure the importance of specific elements. The value of the corresponding weights could give one way of measurement, following the idea of 'the higher the corresponding weight - the more important the element'. We name this approach **WeightPruning** and apply it after training for a few iterations. This is closely related to the previously introduced perceptron pruning, but it is not limited to a perceptron. Instead, it is defined for general MLPs. A WeightPruning $WP(m,n)$ with an initial **B** of size $|\mathbf{B}| = m$ is performed as follows: Define n to be $n < m$. We train the network with a mapping defined by **B**. After training the network, we introduce **B*** in such a way that **B*** consists of only those elements of **B** where the respective weights are greater than a margin chosen so that $|\mathbf{B}^*| = n$. With that, we believe that **B*** will provide superior performance as it contains the most important frequencies of the signal that is to be reconstructed.

Recently, Wang *et al.* (2021a) showed that for the architecture search task, determining an element's relevance by the value of its corresponding weight is not optimal. Instead, they propose another way of measurement: to exclude individual elements and evaluate the performance of the resulting network. We name this approach **UnitPruning** and apply it after training for a few iterations. A UnitPruning $UP(m,n)$ with an initial **B** of size $|\mathbf{B}| = m$ is done as follows: Each Fourier parameter $b_i \in \mathbf{B}$ has its associated weight value w_i. By setting $w_i = 0$, we exclude the Fourier parameter and can calculate the resulting loss L_i. We then measure the importance of a parameter by the size of the resulting loss L_i. If the performance drops by a considerable amount, we say that the parameter is important, and a minor performance drop means that the parameter is unimportant. The Fourier parameters, with the highest L_i, are then removed.

How to Replace the Removed Elements?

After elements of the Fourier embedding have been removed, the task is now to replace them. We propose a replacement strategy that is built on the observation from Tancik *et al.* (2020); Benbarka & Höfer *et al.* (2022), that the most critical component for good performance is the standard deviation of the Fourier embedding rather than the actual values of the parameters. Assuming now that the reason for removing the elements in

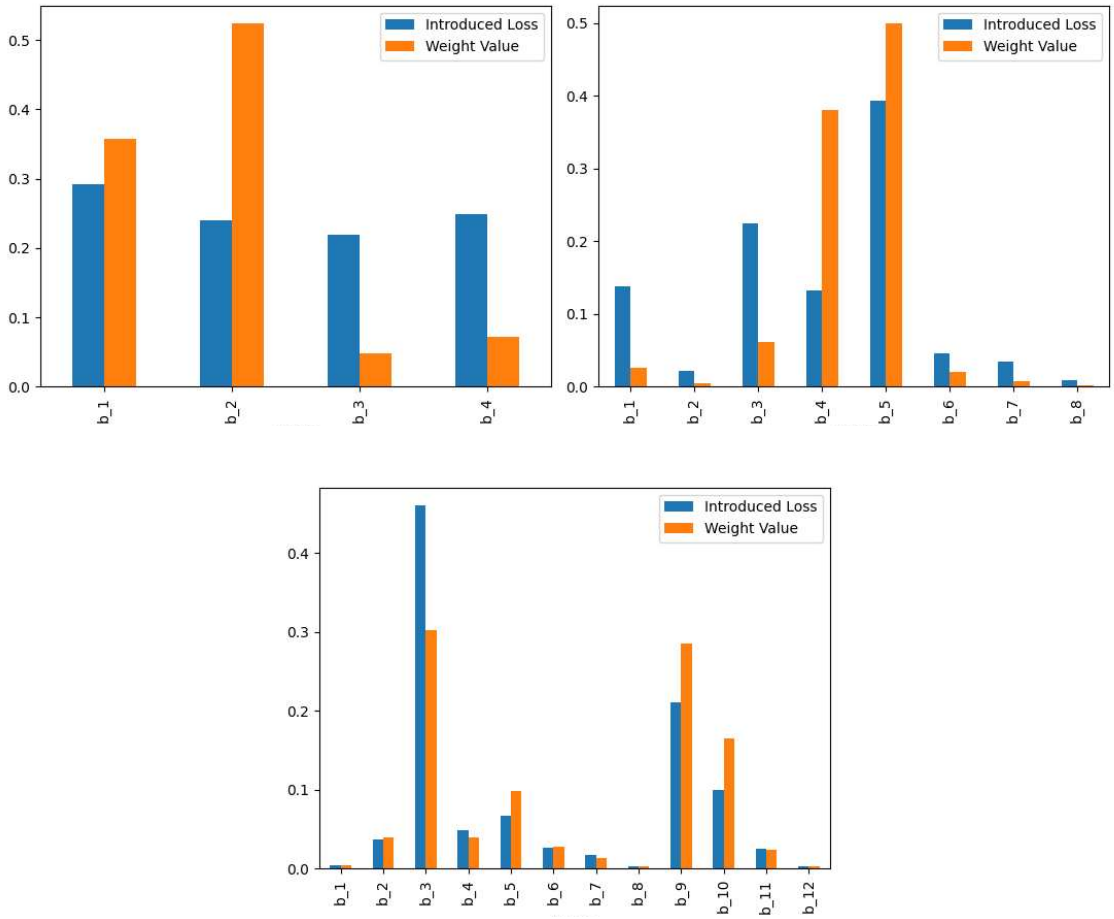


Figure 3.13: The weight value and introduced loss of an element b_i from the Fourier embedding matrix $\mathbf{B}$, after applying softmax. The magnitude of the accociated weight for some b_i does not necessarily agree with its introduced loss at convergence.

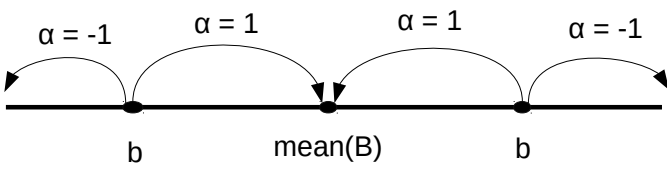


Figure 3.14: Influence of α on the change of some Fourier value b, for two different positions. If $\alpha = 1$, the value b will be moved to the mean, reducing the overall standard deviation of the elements in $\mathbf{B}$. Contrary, if $\alpha = -1$, the overall standard deviation will be increased.

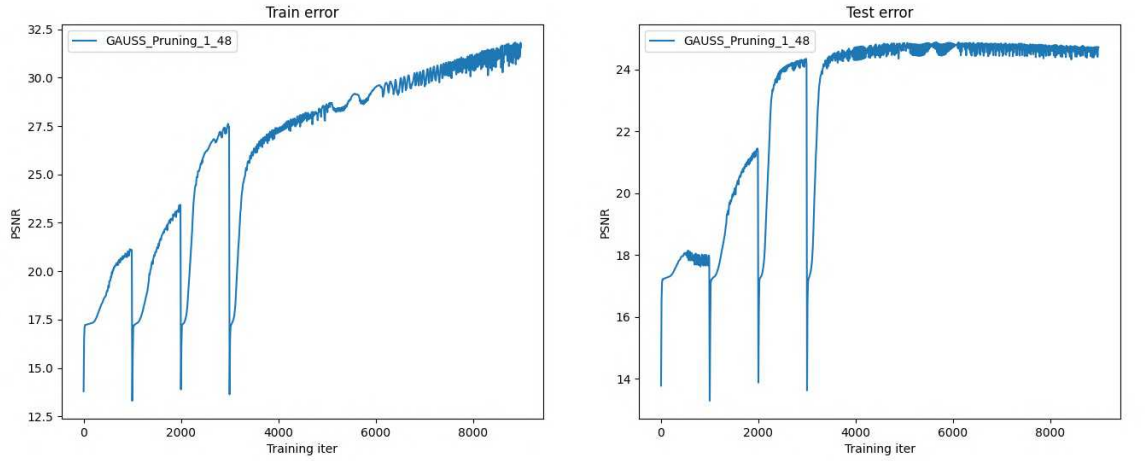


Figure 3.15: The training progress of our iterative method. After 1000 iterations, the Fourier embedding matrix **B** is updated, and the network weights are reset. With each iteration, the quality of the output is increased.

the previous step was the influence on the standard deviation of the whole embedding, we want to replace them with elements that have the opposite influence on the standard deviation. This is done as follows:
Given the Fourier parameters $\mathbf{B} = (b_1, \ldots, b_d)$, and the information that in the previous step the element b_i was removed, the replacement is

$$b_i^{\text{new}} = b_i + \alpha[mean(\mathbf{B}) - b_i]. \tag{3.29}$$

We want to define the parameter α in such a way that it follows the structure of Figure 3.14. For the direction we want, that if $\sigma_{\text{old}} < \sigma_{\text{new}}$ that $\alpha < 0$ and if $\sigma_{\text{old}} > \sigma_{\text{new}}$ that $\alpha > 0$ respectively. In words, this means that if the removed b_i was responsible for a high standard deviation of the Fourier parameters, then the new b_i^* will be closer to the mean and therefore reduce the overall standard deviation. Hence, we define α in the following way:

$$\alpha = s \cdot (\sigma_{\text{old}} - \sigma_{\text{new}}) \frac{1}{|mean(B) - b_i|^\gamma}. \tag{3.30}$$

Here s and γ are hyperparameters, and the fraction on the right side has the purpose of accommodating the closeness of b_i to the mean. Finally, we manually add some restrictions to the value: by limiting $\alpha \in [-1, 1]$, we take care of outliers that would introduce big steps and guarantee that in case of a positive α value, we are not moving away from the mean (which would happen with $\alpha > 2$).

Simplified Version:

Type	init. size	n = 6	n = 8	n = 10	n = 12
No	n	15.23	17.21	17.36	18.55
W.Prune	18	**19.02**	20.16	20.96	18.73
	24	17.94	20.62	20.78	**21.34**
	32	16.67	19.53	**21.08**	21.25
U.Prune	18	**19.02**	18.57	20.73	18.73
	24	17.94	**21.28**	20.67	20.59
	32	15.23	20.0	20.50	21.33

Table 3.4: A network with an initial Fourier embedding matrix of size $|\mathbf{B}| = m$ is initially trained. Then, using the removal step from section 3.4.2, less important elements of $\mathbf{B}$ are removed to yield a size $|\mathbf{B}^*| = n < m$. This table shows the resulting **training** PSNR scores with a standard deviation of **32**.

Type	init. size	n = 6	n = 8	n = 10	n = 12
No	n	9.40	11.59	10.16	11.54
W.Prune	18	**17.03**	17.41	17.24	16.25
	24	16.17	17.80	**17.66**	17.28
	32	13.79	14.47	16.86	17.02
U.Prune	18	**17.03**	**18.57**	16.84	16.25
	24	16.17	17.74	17.35	17.28
	32	12.23	15.40	16.46	**17.56**

Table 3.5: This table shows the **test** PSNR scores from the experiments in Table 3.4.

If the scale s is chosen too big, the resulting update step is

$$\alpha \in \{-1, 1\}. \tag{3.31}$$

This leads to a simplified version of the replacement. Basically, one only needs to calculate the sign of $\sigma_{\text{old}} - \sigma_{\text{new}}$ and receives the α value without the need to adjust the hyperparameters s and γ.

3.4.3 Experiments

We focus on the image regression task. Given an Image I, the aim is to learn a function $f : (x, y) \to (R, G, B)$, that maps each pixel coordinate (x, y) to its corresponding *RGB* value. First, we define the function f to be a multilayer perceptron. As discussed in the previous chapter, we apply a Fourier mapping to the pixel coordinates, which is then fed through the network. We evaluate the images from figure 3.7 with a size of 512×512.

Type	init. size	n = 6	n = 8	n = 10	n = 12
No	n	15.18	16.93	18.03	17.57
W.Prune	18	16.46	19.42	19.66	18.75
	24	15.57	19.62	19.72	20.39
	32	14.33	19.05	18.23	20.50
U.Prune	18	**17.42**	**19.88**	**20.05**	19.21
	24	14.28	19.69	19.72	**20.74**
	32	14.33	19.59	18.67	20.56

Table 3.6: A network with an initial Fourier embedding matrix of size $|\mathbf{B}| = m$ is initially trained. Then, using the removal step from 3.4.2, less important elements of $\mathbf{B}$ are removed to yield a size $|\mathbf{B}^*| = n < m$. This table shows the resulting in **training** PSNR scores with a standard deviation of **48**.

Type	init. size	n = 6	n = 8	n = 10	n = 12
No	n	8.65	9.30	8.53	8.74
W.Prune	18	11.04	12.23	11.38	10.50
	24	**12.08**	**14.34**	**14.24**	13.32
	32	11.65	12.00	12.88	12.64
U.Prune	18	9.37	12.40	12.44	10.87
	24	11.67	13.62	**14.24**	13.13
	32	11.65	13.77	13.43	**13.77**

Table 3.7: This table shows the **test** PSNR scores from the experiments in Table 3.6.

The results are given in PSNR: the peak signal-to-noise ratio (as it is a scaled logarithm of the squared maximum pixel value in the image divided by the MSE, higher values describe a better performance). While we define the training set to consist of every second pixel in the image, the test is the whole image. In general, we use the MSE loss combined with the Adam optimizer.

Experiments on the Pruning Task

In this section, we compare pruning techniques following section 3.4.2. To conduct the experiments, we choose the network architecture to have a depth of 6 and a width of 128. We sample the initial Fourier parameters $\mathbf{B}$ from a normal distribution $\mathcal{N}(0, \sigma^2)$. We intentionally choose the variance σ^2 to be not optimized, such that we can analyze whether the pruning techniques are able to select appropriate elements. Both pruning techniques start with the same sampled $\mathbf{B}$, with an initial size of $|\mathbf{B}| = m$. After training for a few iterations, we use the pruning techniques to choose the most suitable subset $\mathbf{B}^*$

		Train			Test		
		n = 24	n = 32	n = 48	n = 24	n = 32	n = 48
	Basic	21.01			20.34		
	PE	23.90	25.18	25.16	21.35	21.17	21.34
non-optimal initialization	No replacement	21.1	21.59	22.51	13.91	13.47	13.38
	Barf	22.2	25.18	25.75	17.25	19.40	20.64
	Ours (3.4.2)	25.63	**27.76**	**28.88**	**22.64**	**22.66**	**21.83**
	Gaussian 4 it.	24.39	23.72	23.09	17.71	16.27	14.31
	Gaussian 15 it.	**25.86**	26.00	26.88	21.41	19.83	18.04

Table 3.8: Train and test PSNR scores of the whole procedure using different Fourier embedding sizes n and replacement strategies. We compare our replace strategy from section 3.4.2 with a gaussian replacment, the progressive training strategy from Barf (Lin *et al.* (2021)). and for comparison, the results one would obtain without our iterative method. Furthermore, we compare it with common approaches, basic embedding and postional encoding.

of $\mathbf{B}$ with size $|\mathbf{B^*}| = n < m$. Finally, we compare the performance when training with the new Fourier embedding matrix $\mathbf{B^*}$ and also compare it to the results one would achieve when directly sampling a matrix of size n from $\mathcal{N}(0, \sigma^2)$.

To demonstrate that the pruning techniques yield a different solution, we have a closer look at Figure 3.13. After training, we compare the elements of the Fourier embedding matrix B. The sizes of $\mathbf{B}$ are 4, 8 and 12. The orange bars represent the weight value of the neural network that is associated with the element b from $\mathbf{B}$. The blue bars are the introduced losses that we get if we set the associated weight value to 0. For comparability, we used the softmax function on the original values. While, in general, it looks like there is a relation between the two values, the order of the elements is different. Therefore, it is necessary to conduct more experiments on the performance of the pruning techniques.

In the tables 3.4, 3.5, 3.6 and 3.7, we can see that for both the train and test data, using any pruning technique outperforms a direct sampled $\mathbf{B}$ of size n. Both pruning techniques show comparable performance. If the gap between the two parameters m and n is too big, both tend to lose quality. Contrary, choosing m close to n would reduce the number of elements to choose from. Overall, we see slightly better performances of UnitPruning, aligning with the results in Wang *et al.* (2021a). Therefore, we continue with UnitPruning in the following experiments.

For each iteration step, it is necessary to do some initial training. To determine the optimal number of training iterations, we do the following experiment: We take different networks with some Fourier embedding matrix $\mathbf{B}$. In Figure 3.16 the x-axis determines the number of iterations until we apply our UnitPrune. After the pruning, we train the network for a fixed length and compare the resulting PSNR scores. Here, the same PSNR

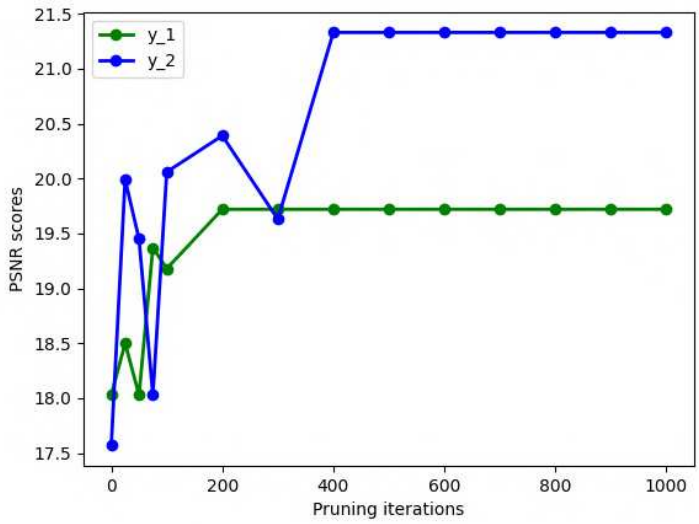


Figure 3.16: Experiments on the number of iterations before applying the pruning techniques. A UnitPruning is applied after training for the given amount of iterations. Similiar PSNR scores mean, that the pruned elements are the same.

scores imply that the resulting Fourier matrix **B*** are the same, which means that after the latest 300 iterations, the pruning method yields no changes. For our experiments, we then used 1000 iterations; as for the task of image regression, the amount of time is considerably small.

Experiments on the Whole Procedure

In addition to the experiments in the previous section, we now replace the removed elements of **B**. In this experiment, we update 5 Fourier parameters with 1000 iterations steps between the update steps. We compare our approach described in section 3.4.2 with the original performance and a replacement when simply sampling from a normal distribution with the same standard deviation. Also, we compare it to the progressive training strategy we introduced in section 3.3.1, where the high-frequency activations of the encodings are hidden at the beginning of the training and are gradually allowed during the training. All of the upper methods are built on top of a poorly sampled Fourier embedding. We compare the results to a basic embedding that maps the input coordinate to a sinusoidal representation with a fixed size and the commonly used positional encoding.

In Table 3.8, one can see that an initially bad Fourier embedding matrix **B** can be modified during training by our automatic adjustment, such that after only a few iterations, the resulting PSNR values are appropriate. Also, the sampling from a Gaussian distribu-

	NeRF
Basic	23.16 ± 0.90
Positional encoding	24.81 ± 0.88
Gauss initial	20.22 ± 0.89
Gauss iterated	25.17 ± 0.92

Table 3.9: Resulting PSNR scores of the view synthesis task using a 'tiny' NeRF. Our iterative method can transform a poorly chosen initial Gauss embedding into a well performing one.

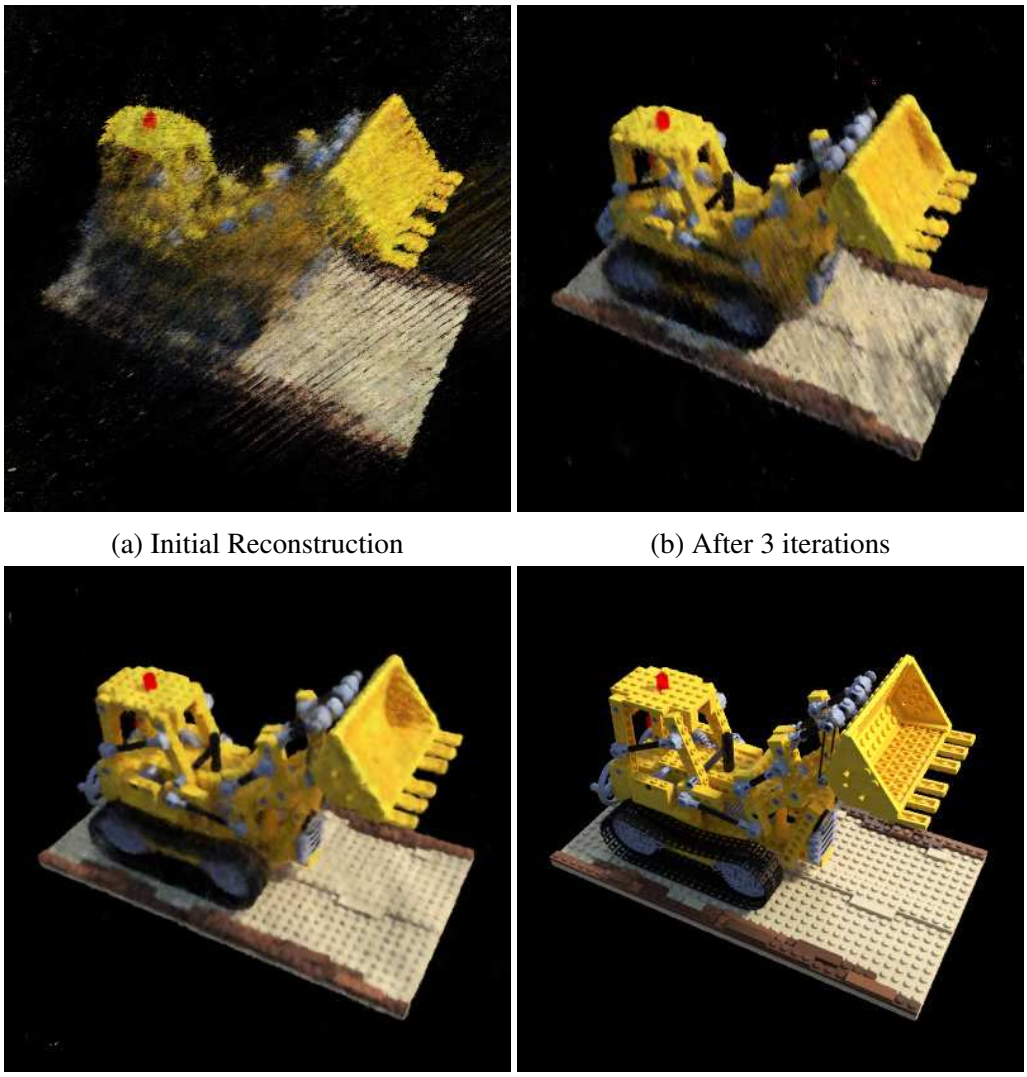
tion gives a good performance, with the difference that it needs way more iterations until the performance saturates. Additionally, our approach yields the best test PSNR scores. It should also be remarked that the replacement is dependent on the initially chosen standard deviation in the Gaussian case. Hence a bad selection at the beginning would badly influence the replacement.

To guarantee that the network is not stuck at a local minimum, we reset all weights of the network after the replacement to guarantee the inclusion of the new Fourier parameters. As shown in Figure 3.15, this results in jumps of the training procedure, yielding a performance increase with each iteration. To prevent the network from losing all the information from the previous training steps, we explored the possibilities of skipping the weight reset, adding random noise, or resetting the weights of the first layer. Experiments showed that none of these approaches gave satisfying results. Still, it is possible to analyse possibilities further to save some effort from the previous training steps.

Experiments on the Novel View Synthesis Task

To test whether our method is applicable to novel view synthesis, we chose a 'tiny' NeRF where we removed view dependence and hierarchical sampling. Generally, a NeRF takes as input a set of 2D images and tries to recreate a 3D representation of the scene. In this scenario, we are given three-dimensional input coordinates (x, y, z) and map them to (R, G, B, σ), where σ is the volume density.
The corresponding MLP has a width of 256 and a depth of 4. We trained the model for 50.000 iterations with the Adam optimizer and individually optimized learning rates. For evaluation, we take the mean PSNR scores of unseen test images from different viewing angles.

In Fig. 3.17 the qualitative improvements of our iterative method can be seen. Starting with a non suitable Gaussian Fourier initialization, we receive after a few iterations satisfactory results. Quantitative results (mean PSNR scores and standard deviation) can be found in Table 3.9, where we outperform traditional approaches, namely the positional encoding and the basic embedding.



(a) Initial Reconstruction (b) After 3 iterations

(c) After 7 iterations (d) Ground truth

Figure 3.17: View synthesis performance using a 'tiny' NeRF. An initial poorly chosen Fourier embedding matrix produces poor reconstructions (a). We gradually increase the reconstruction by using our iterative method (b,c).

3.4.4 Conclusion

This work introduced an iterative method to adjust the Fourier parametrization, consisting of a remove and replace step. We introduced techniques for the steps and found reasonable working solutions for both. Our method can be tested with any initial Fourier embedding and is designed to account for poorly chosen parametrizations. With this, the need to search for an optimal Fourier embedding is redundant. In our work, we focused on the Fourier embedding itself, using the image regression task and view synthesis task as application. Future work can analyze whether our method can be applied to other tasks, such as sounds, occupancy and others.

Chapter 4

From Object Detection to Instance Segmentation

4.1 Technical Introduction

Many researchers have investigated a wide range of sub-problems in the varied and well-known field of computer vision. We will introduce the tasks of image classification, object detection and image segmentation. After a detailed literature review, we will introduce our work 'FourierMask: Instance Segmentation using Fourier Mapping in Implicit Neural Networks' by Riaz *et al.* (2022). It falls under the category of instance segmentation by utilizing an implicit mask predictor.

4.1.1 Task Definition

Image Classification The process of assigning a single label to a picture is known as object classification. Typically, the input is a cropped image with only one foreground object visible, and the output is a label for that image. See the left image of Fig. 4.1. With the rise of convolutional neural networks, architectures like VGG16 and ResNet have demonstrated state-of-the-art performance in the classification task on the ImageNet dataset introduced in Deng *et al.* (2009). While this is a fundamental problem in computer vision, classification is not covered in this work.

Object Detection One can think of object detection as object categorization plus localization. An image with any number of objects at any scale serves as the input, and the result is a set of bounding boxes. Each bounding box gives a label to identify the class (or kind) of an object in the box and defines a region in the image. 3D object detection, where the bounding box tries to identify the item's full location in 3D space, and 2D object detection, where the bounding box is defined just in the image plane, are being worked on. Since 2D object detection is the primary subject of this study, we will call it "(object) detection." For an example, input image and bounding box output, see the middle image of Fig. 4.1.
Object detection is of particular interest to us as it aids in pose estimation. Object de-

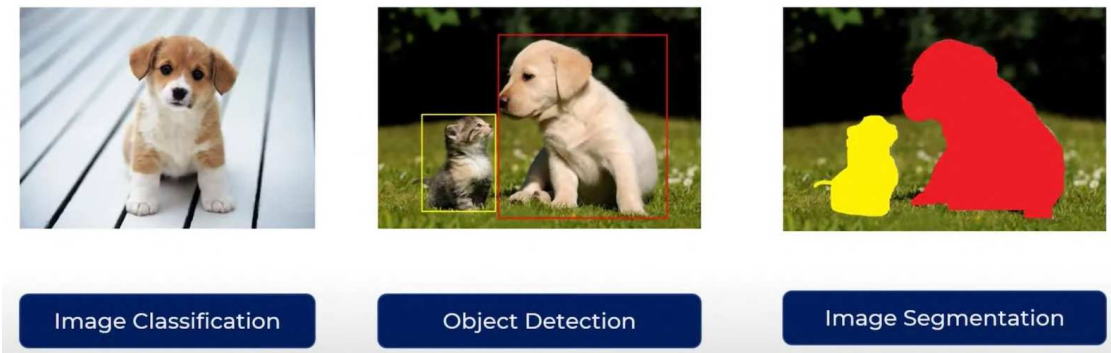


Figure 4.1: The difference between classification, detection and segmentation, credit: Codebasics (2020).

tection algorithms differ from classification algorithms in that they attempt to locate the object of interest by drawing a bounding box around it. Furthermore, there may not always be just one bounding box drawn for object detection. There might be a number of bounding boxes representing different objects of interest within the image, which are not known beforehand. This problem cannot be solved by appending a fixed fully connected layer on top of a typical convolutional network, as the output layer's length is variable rather than constant since the number of the items of interest is not fixed. A simplistic solution to this issue would be to select distinct areas of interest from the image and use a CNN to determine whether the object is present in that area. The issue with this method is that the objects of interest could be located differently in the image's space and have various aspect ratios. Therefore, one would have to choose a large number of locations, which could result in excessive computing times.

Consequently, techniques like R-CNN by Girshick *et al.* (2014), YOLO by Redmon *et al.* (2016), and others have been introduced to handle the search of finding the occurrences of the present objects fast and reliably. Girshick *et al.* (2014) suggested an approach where they use selective search to extract just 2000 regions from the image, and he dubbed them region proposals to get over the issue of selecting a large number of regions. Therefore, one may now deal with 2000 regions rather than trying to categorize a large number of regions. These region proposals are then fed through a convolutional neural network to produce a feature vector. Furthermore, the algorithm predicts four offset values to increase the precision of the bounding box, as well as the presence of an object within the region proposals. The R-CNN networks still suffer from problems such as a lack of speed because one has to process all of the 2000 regions per image and hence it cannot be implemented in real-time. Also, the selective search algorithm is fixed and therefore, no learning is happening in this stage, which could lead to a lousy region proposal generation. Some drawbacks have been solved by Fast R-CNN (Girshick (2015)). The main difference is that instead of feeding the region proposals to the CNN, the input

image is directly fed through a CNN to yield a feature map from which region proposals are then identified. As there is no need anymore to put the 2000 region proposals through the CNN for all images, it speeds up the inference by a significant amount. Still, the inference is not in real-time, as the selective search algorithm is time-consuming. To achieve real-time performance, Faster R-CNN (Ren *et al.* (2015)) was introduced, which eliminates the need of the selective search algorithm. Instead, the region proposals are predicted with a separate network which enables real-time object detection.
While the R-CNN-based detection algorithms make use of region proposals, we want to introduce YOLO, a network that only looks at parts of the image with a high probability of containing an object. The bounding boxes and class probabilities for these boxes are predicted by a single convolutional network. In order for YOLO to function, we first divide a picture into a grid and then take multiple bounding boxes for each grid. The network outputs a class probability and offset values for the bounding box for each bounding box. The object is identified within the image by selecting the bounding boxes with the class probability over a threshold value. YOLO is considerably faster than other object detection methods. The YOLO algorithm's weakness is that it has trouble detecting small objects in the image; for instance, it might have trouble spotting a flock of birds. The algorithm's spatial limitations are to blame for this. There are multiple works on top of YOLO, which can be found in the literature and are called YOLO v1, ..., and YOLO v7, that are improved versions of the initial work from Redmon *et al.* (2016).
Besides the now-introduced methods, there are more that can be found in the literature; however, as the R-CNN and YOLO networks are the most influential ones, we only introduced them here.

For applications that need to be done in real-time such as a safety system on a vehicle, it is important that the detectors have a high framerate. With the usage of a modern GPU such as the Nvidia RTX 3060, this can be achieved with most state-of-the-art detectors. In the case that one only has a CPU, things get more complicated. One has to look for specific network designs, such as the MobileNet family. The orange bar in figure 4.2 represents the framerate on an i5 CPU, yielding significantly lower values compared to the GPU. Still, the usage of mobilenet backbones enables a framerate that comes close to 15. In other scenarios, for example, in the case of detection on a drone or a land machine, one needs a mobile GPU, for example, the Nvidia Jetson AGX Xavier, where we conducted another benchmark of a subset of the previous detectors. Following figure 4.3 it is possible to receive framerates over 20 fps.

Image Segmentation Related to object detection, image segmentation produces a more precise identification of the image. The input is similar to the input of object detection, but the result is a class label for each pixel of the input image. See Figure 4.1's right column for an illustration. A special type of image segmentation is instance segmentation, which focuses on instances of objects and demarcates their boundaries. In this instance, the dog's and cat's pixels are classified as belonging to the respective class, while all other pixels are classified as "background." We will focus on instance segmentation methods

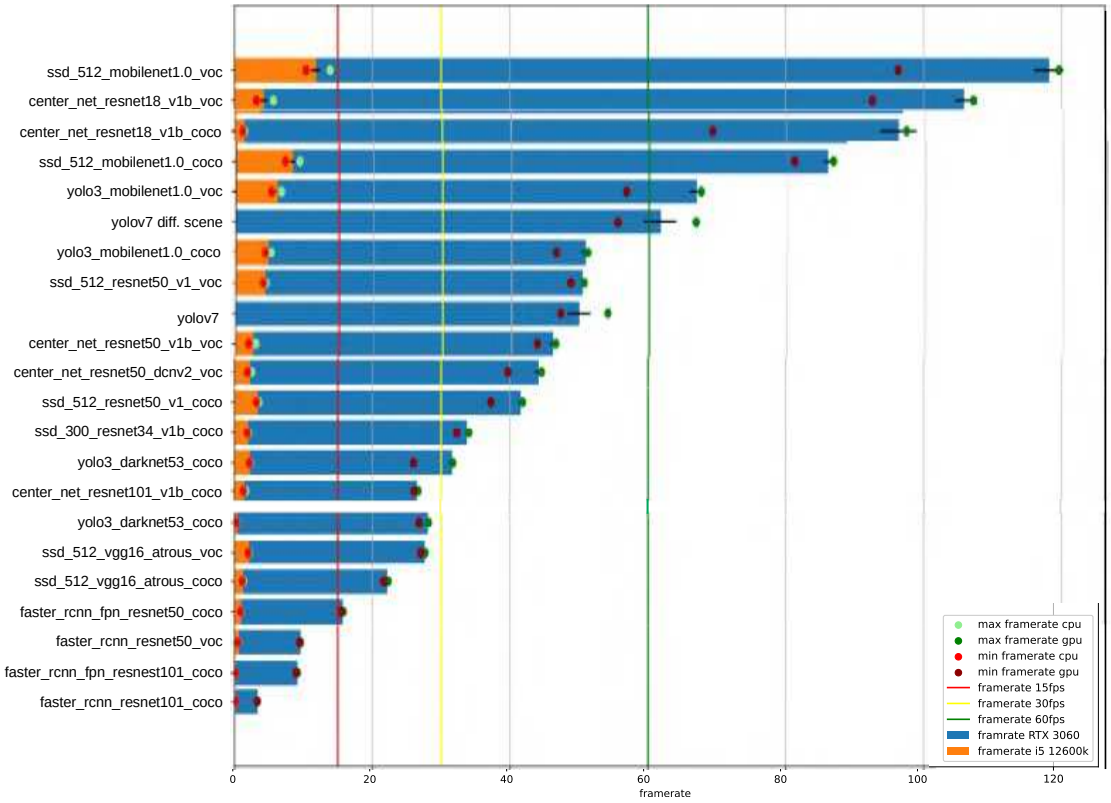


Figure 4.2: A benchmark on the framerate of common object detectors. The orange bar represents the framerate on a CPU, and the blue bars stand for the framerate utilizing a GPU respectively. The fastest detectors can reach a framerate of over 100 fps.

in the following.

In *two-stage* instance segmentation, the network first detects (proposes) the objects and then predicts a segmentation mask from the detected region. The baseline method for many two-stage methods is Mask R-CNN (He *et al.* (2017)). Faster R-CNN introduced by Ren *et al.* (2015) was given an additional mask branch resulting in Mask R-CNN, which created a binary mask that distinguished between the foreground and background pixels in a region of interest. The mask IoU was regressed, and the mask scoring R-CNN had a network block to learn the quality of the anticipated instance masks in Huang *et al.* (2019). ShapeMask, which is described in Kuo *et al.* (2019), calculated the shape from bounding box detections using shape priors, and then it was further refined into a mask by learning instance embeddings. Focusing on crucial pixels and reducing noise were made possible by CenterMask's use of a spatial attention-guided mask on top of FCOS's object detector introduced in Lee and Park (2020). PointRend addressed instance segmentation as a rendering issue which was introduced in Kirillov *et al.* (2020). A fully linked MLP was used to sample uncertain points from the feature map and its

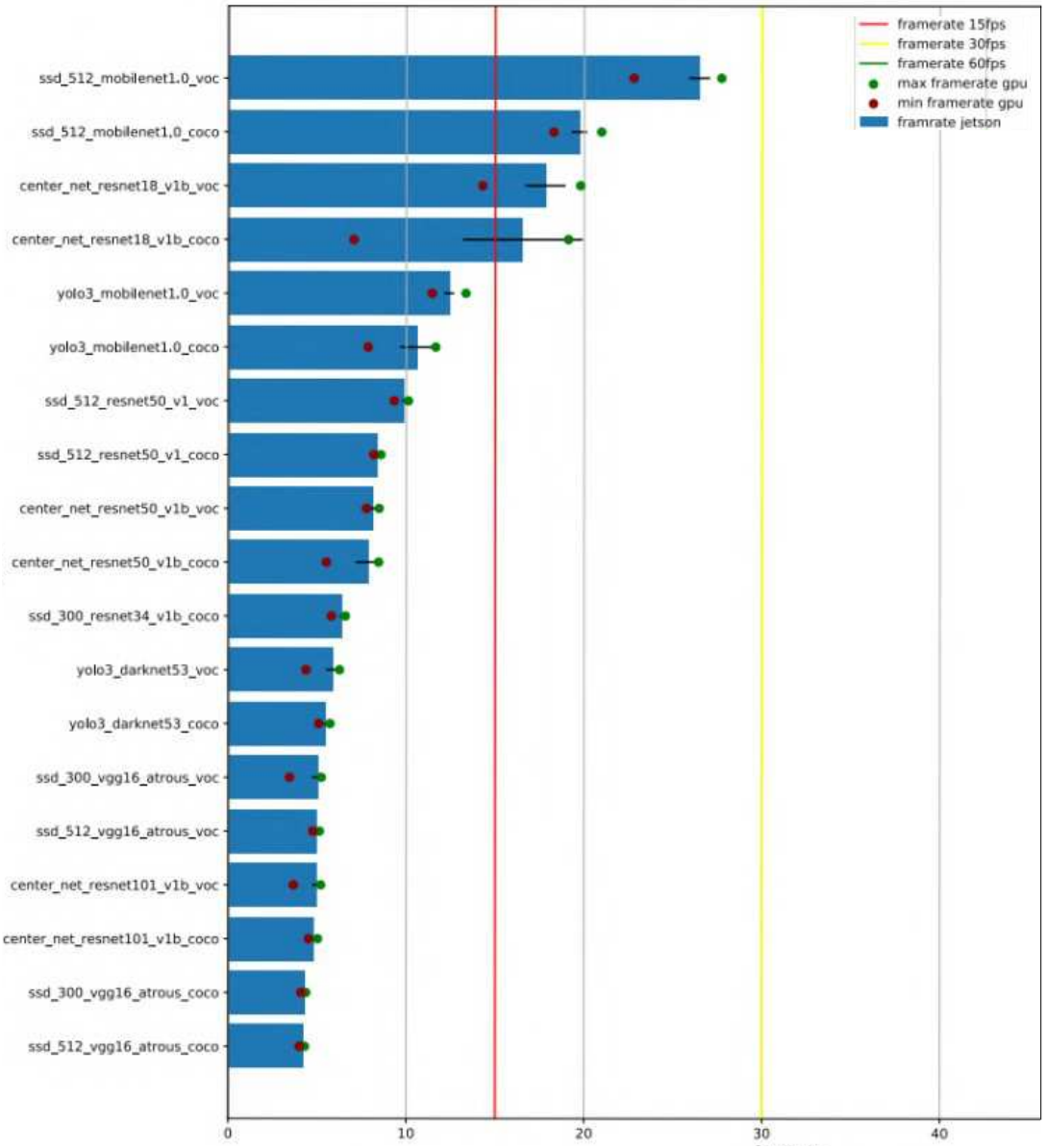


Figure 4.3: A benchmark on the framerate of common object detectors that define a subset of the introduced detectors in figure 4.2. For this experiment, we utilized an Nvidia Jetson AGX Xavier.

fine-grained features from a higher-resolution feature map in order to forecast extremely sharp object boundaries. PolyTransform by Liang *et al.* (2020) employed a polygon representation of masks rather than a binary grid representation. High accuracy is achieved with these approaches, but they are typically slower than one-stage procedures.

One stage instance segmentation methods can predict in a single shot, without using any proposed regions/bounding boxes as an intermediate step, the instance masks. To estimate masks at real-time speeds, YOLACT used a linear combination of prototype masks and mask coefficients for each instance. Embedmask, introduced in Ying *et al.* (2019), also used embedding modules for pixels and mask suggestions. Using the object's keypoints, in Zhou *et al.* (2019a) where they introduced ExtremeNet, estimated the contour (octagon) around the object. Similar to how Polarmask from Xie *et al.* (2020) predicted a contour from a center point, Tian *et al.* (2019) introduced FCOS and used the polar representation. Dense RepPoints introduced in Yang *et al.* (2019) uses a massive collection of points to depict object boundaries. The inverse fast Fourier transform (IFFT) was used by FourierNet in Riaz *et al.* (2021) to create a contour around an object represented by polar coordinates. In order to forecast the outlines, the network learned the Fourier series coefficients.

Before we introduce the task of object pose estimation, we are going to propose FourierMask, a method for segmentation that utilizes INRs for this task.

4.2 FourierMask: Instance Segmentation using Fourier Mapping in Implicit Neural Networks

This section introduces our work in Riaz *et al.* (2022), where we utilized implicit neural representations for the instance segmentation task, which we named FourierMask. It utilizes a Fourier mapping to the coordinate locations and takes the mapped features as inputs to an implicit neural representation which is then able to generate an accurate segmentation mask. FourierMask learns to predict the FM coefficients for a specific instance and thus adapts the FM to a specific object. FourierMask can now be generalized to predict instance segmentation masks from raw images. Because implicit functions are continuous in the domain of input coordinates, we show how we can generate higher-resolution masks during inference by subsampling the input pixel coordinates. Furthermore, we train an MLP renderer *extended FourierMask* on the uncertain predictions of FourierMask and show that it significantly improves mask quality. FourierMask matches the baseline Mask R-CNN at the same output resolution on the MS COCO dataset and outperforms it at higher resolutions.

Instance Segmentation Using Implicit Neural Representations

There has been a shift from classical approaches towards deep learning methods in the past decade. The availability of real and synthetic data and high computation capabilities have made it possible to use these 'black box' models on highly complex and critical problems. For instance, segmentation CNNs have been used most commonly in recent years. Instance segmentation masks can be generated by classifying pixels in a region of interest as either foreground or background, such as Mask R-CNN by He *et al.* (2017).

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

Figure 4.4: Comparison between Mask R-CNN by He *et al.* (2017), PointRend by Kirillov *et al.* (2020) and FourierMask.

The performance of these methods is generally the best, but they are slower and require a lot more computation. It is possible to predict the contour points around the edge of an object (Yang *et al.* (2019), Xie *et al.* (2020)). Despite being faster, such methods fall short of pixel-wise classification methods in terms of performance. Some methods attempt to encode the mask contours in a compressed representation (Xu *et al.* (2019), Riaz *et al.* (2021)). Although these mask representations are compact and meaningful, they fall short of the superior capabilities of pixel-wise methods. The general shape of segmentation masks is held by the low-frequency components of the Fourier series, while the edges of the mask are held by the high-frequency components. As a result, our representation is meaningful and can be compressed based on the use case.

FourierMask is an *implicit neural representation*. As we know from chapter 3, for the task of image regression, *implicit representations* can be thought of as learnable functions characterized by neural networks that map a pixel coordinate to its corresponding RGB value. We further showed that applying a Fourier mapping to the coordinates as a pre-step allows the implicit networks to learn high-frequency details in images and 3D scenes. FourierMask draws inspiration from this work and applies this knowledge to the task of instance segmentation. In comparison to traditional representations, implicit representations have the advantage of learning and reconstructing fine details, which traditional representations find difficult to do in such small models. We can sub-sample the pixel coordinates to generate higher-resolution masks during inference because implicit functions are continuous in the domain of input coordinates.

In this work, our contributions are as follows:

- We develop FourierMask, which can replace any mask predictor that uses a region of interest (ROI) to predict a binary mask. It is fully differentiable and end-to-end trainable.

- We show that implicit representations can be applied to the task of instance segmentation. We achieve this by learning the coefficients of the Fourier mapping of a particular object.

- As implicit functions are continuous in the domain of input coordinates, we show that we can sub-sample the pixel coordinates to generate higher-resolution masks during inference. These higher-resolution masks are smoother and improve the performance on MS COCO.
- We verify and illustrate that the rendering strategy from PointRend (Kirillov *et al.* (2020)) brings significant qualitative gains for FourierMask. Our renderer *FourierRend* improves the mask boundary of FourierMask significantly.

4.2.1 Method

The Head of FourierMask to Predict Fourier Coefficients

This section explains the network architecture of FourierMask. We employ Mask R-CNN (He *et al.* (2017)) as our baseline model. We use a ResNet (He *et al.* (2016)) backbone pre-trained on the ImageNet dataset (Deng *et al.* (2009)), with a feature pyramid network (FPN) (Lin *et al.* (2017b)) architecture. Following Mask R-CNN, we use a small region proposal network (RPN), which generates k proposal candidates from all feature levels from the FPN. To generate fixed-size feature maps from these proposal candidates, we use an ROI Align (He *et al.* (2017)) operation. This produces a (k,d,m,m) sized feature map for the mask head, where m is the fixed spatial size after the ROI Align operation, and d is the number of channels. The structure of the head is shown in figure 4.7. We apply four convolutions consecutively, each with a kernel size 3×3 and a stride of 1. Then we apply a transposed convolution layer with $2c$ number of filters, which generates a spatial volume of size $2c \times 2m \times 2m$. We call this feature volume $\mathbf{W}$, which holds $2c$ Fourier coefficients for each spatial location.

Generate Fourier Features from the Fourier Coefficients

In this section, we explain how to obtain the Fourier features from the coefficients $\mathbf{W}$. We use the integer lattice mapping as our Fourier embedding, which we introduced in section 3.3.1

$$\gamma(\mathbf{x}) = \begin{pmatrix} \cos(2\pi\mathbf{B}\cdot\mathbf{x}) \\ \sin(2\pi\mathbf{B}\cdot\mathbf{x}) \end{pmatrix}. \tag{4.1}$$

Here $\mathbf{x} \in \mathbb{R}^{p\times 2}$ are the pixel coordinates, normalized to a value in the range of $[0,1]$ and p are the total number of pixels in the image. Since sine and cosine have a period of 2π, by normalizing the pixel coordinate $\mathbf{x}$ to the range of $[0,1]$, we ensure that one complete image lies in a period of 2π. Images are not periodic signals, but since they are bounded by the image resolution, we can safely apply our method to predict 2D binary masks as they can be thought of as their existing continuous extension over their input bounds. $\mathbf{B} \in \mathbb{Z}^{2\times c}$ is the integer lattice matrix which holds the possible combinations of harmonic

frequency integers of Fourier series for both dimensions in the image. As we are given a dimension of 2, formula 3.21 tells us that the number of coefficients from the embedding is

$$c = (N+1)(2N+1) - N, \tag{4.2}$$

where N is the frequency of the embedding, which is a hyperparameter. Fourier Features are generated as follows:

$$\mathbf{FF(x, W)} = \gamma(\mathbf{x}) \circ \mathbf{W}, \tag{4.3}$$

where $\circ$ is the element-wise (Hadamard) product, $\mathbf{W} \in \mathbb{R}^{p \times 2c}$ is the weight matrix predicted by the FourierMask. Note that we flatten the spatial dimension of the prediction beforehand ($p = 2m \times 2m$).

Basic FourierMask: Predict Binary Mask from Fourier Features

Let $\boldsymbol{ff}_i$ be the i_{th} column of $\mathbf{FF}$; then the binary mask $\mathbf{y}$ is defined as

$$\mathbf{y}(\mathbf{x}, \mathbf{W}) = \phi\left(\sum_{i=1}^{2c} \boldsymbol{ff}_i\right). \tag{4.4}$$

Here ϕ is the sigmoid activation function, which we use to bound the output between 0 and 1. Note that $\mathbf{y}(\mathbf{x}, \mathbf{W})$ can be interpreted as an implicit representation with a single perceptron because it is a linear combination of Fourier features followed by a non linear activation function.

Fourier Features based MLP

As shown by Sitzmann *et al.* (2020b) and Tancik *et al.* (2020), implicit representations can learn to generate shapes, images and more from input coordinates very effectively. Following the work from Tancik *et al.* (2020), we saw that Fourier mapping of input coordinates lets the MLP learn higher frequencies and consequently generate images with finer detail compared to MLPs without Fourier mapping. Furthermore, Sitzmann *et al.* (2020b) showed that using periodic activation functions works better compared to ReLU in implicit neural networks. We employ an MLP with sine activation functions, in which Fourier features (FF) are the input and mask $\mathbf{y}'$ is the output. We have three hidden layers (Siren layers), each with 256 neurons. The MLP has a single output neuron, on which we apply a sigmoid function to bind it between 0 and 1. The Fourier features (FF) are generated by the equation (4.3), and they are parameterized by coefficients $\mathbf{W}$ learned by the network and therefore adapted for a specific input ROI. Coordinate-based MLPs encode the information of one particular image or shape, but by parameterizing them with learned Fourier coefficients $\mathbf{W}$ of each object, we can generalize them to generate a binary mask of any object.

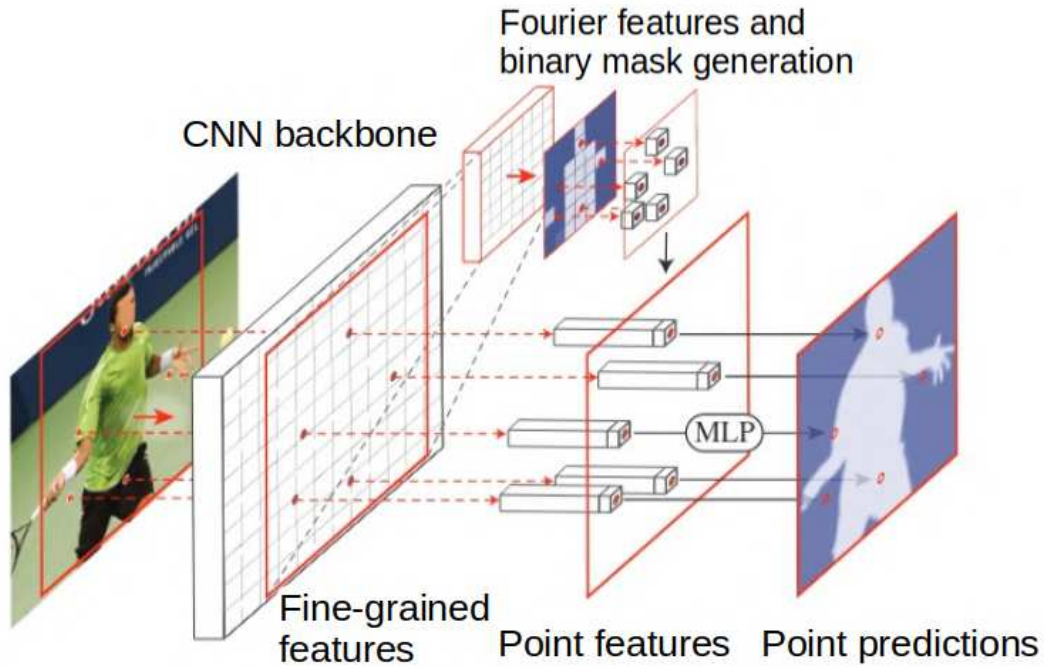


Figure 4.5: Extended FourierMask applied to instance segmentation. The CNN backbone first generates fine-grained features. Region proposals of these features are fed through the FourierMask head to generate the Fourier features and the binary mask. The unsure elements in the binary masks (pixels where the value is close to 0.5) are fed through an additional MLP, in the sense that the MLP receives interpolated features of 1) the fine-grained features and 2) Fourier features as input. This illustration is adapted from Kirillov *et al.* (2020).

4.2.2 Extended FourierMask - MLP as a Renderer

For generating boundary-aligned masks, we used a renderer MLP which specialized only on the uncertain regions of the mask predicted by equation (4.4). We adopted the rendering strategy from PointRend by Kirillov *et al.* (2020) and made the following modification in the point head (figure 4.6). Rather than sampling coarse mask features in the mask head, we sample the Fourier features (**FF**) from equation (4.3) at uncertain mask prediction coordinates (the locations where the predictions are near 0.5). Fourier features **FF(x, W)** make extended FourierMask an implicit MLP since its input is a function of input coordinates **x**, and therefore, we can take leverage from its implicit nature, as discussed before. We concatenate these Fourier features and fine-grained features (from the 'p2' level FPN feature map). We replace the mask predictions from equation (4.4) (coarse predictions), with the fine-detailed predictions from extended FourierMask. Consequently, we replace uncertain predictions at the boundary with more accurate pre-

dictions of the renderer, resulting in crisp and boundary-aligned masks. An illustration can be found in Figure 4.5.

Training and Loss Function

We concatenate the output $\mathbf{y}$ from equation (4.4) and $\mathbf{y}'$ from the MLP and train both masks in parallel. By training the output $\mathbf{y}$, we learn the coefficients $\mathbf{W}$ of a Fourier series in their true sense. We need these coefficients because we assume that the input for the MLP are Fourier features. We use IoU loss for training the binary masks defined as:

$$\textbf{IoU loss} = \frac{\sum_{i=0}^{N} \min(y_{p_i}, y_{t_i})}{\sum_{j=0}^{N} \max(y_{p_j}, y_{t_j})} \tag{4.5}$$

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.

4.2.3 Experiments

For all our experiments, we employ a Resnet 50 backbone with a 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, and we predict only one mask per ROI. For the baseline, we trained a Mask R-CNN, and PointRend by Kirillov *et al.* (2020) with class agnostic masks.

Spectrum Analysis on MS COCO

To validate that the Fourier Mapping (equation (4.4)) works, for instance, in mask prediction, we performed a spectrum analysis on the MS COCO training dataset. Along with verifying our method, this analysis gave us insight into the optimal number of frequencies for the dataset. We performed this experiment by applying a fast Fourier transform

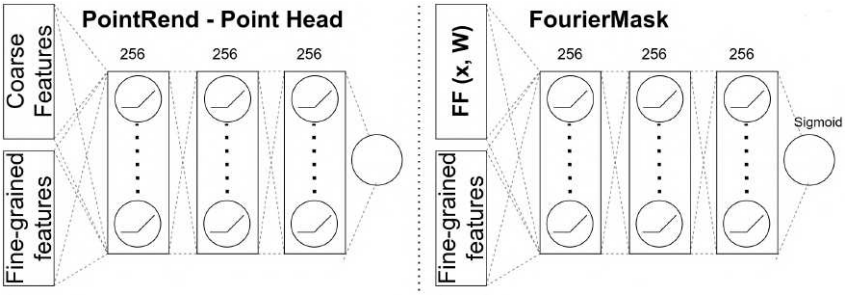


Figure 4.6: Difference between point head from PointRend and extended FourierMask.

on all the target object masks in the COCO training dataset. This Fourier transform gave us the coefficients of a Fourier series, which hold the same meaning as the coefficients prediction **W** of FourierMask. Firstly, we sampled only the lower frequency coefficients of the Fourier series and reconstructed the object's mask by applying equation (4.4). We did this for all the objects' masks in the COCO training set and evaluated the IoU loss of the reconstruction compared to the target. Then we incrementally added higher frequency coefficients and repeated the above-mentioned procedure until we reached the maximum number of frequencies. Figure 4.8 shows the mean IoU loss at various frequencies. It can be seen that the loss decreases exponentially. We choose the maximum frequency as 12 since it has a low enough reconstruction loss and fits comfortably in our GPU memory. Figure 4.9 illustrates a visual comparison between the ground truth and reconstructions using the varying number of frequencies.

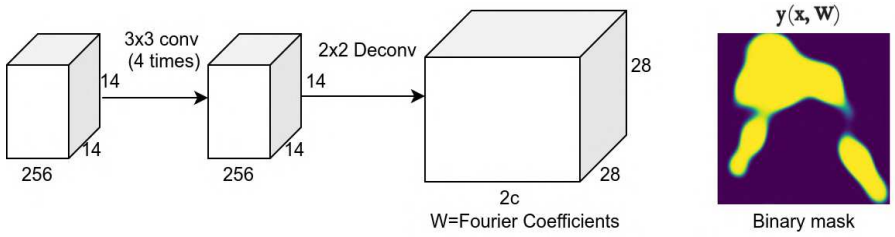


Figure 4.7: FourierMask head architecture for a ROI Align size of 14x14. The network predicts Fourier coefficients **W** for each location in the feature map.

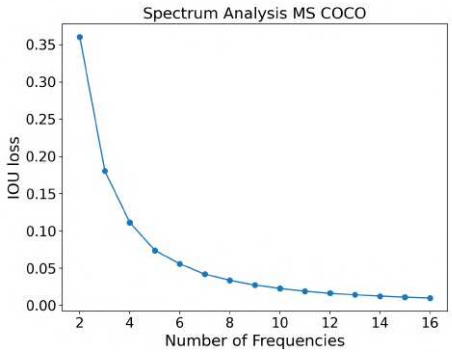
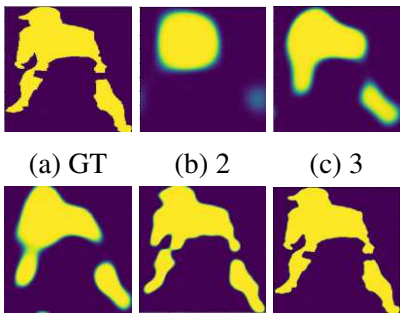


Figure 4.8: IOU vs Frequencies.



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

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

Figure 4.9: The ground truth vs its reconstructions at various frequencies.

Number of Frequencies

To validate our experiment from the previous section, we trained a FourierMask with a similar configuration. Rather than predicting a set of coefficients for each pixel, we modified the architecture to predict a single vector for the whole image. We applied strided

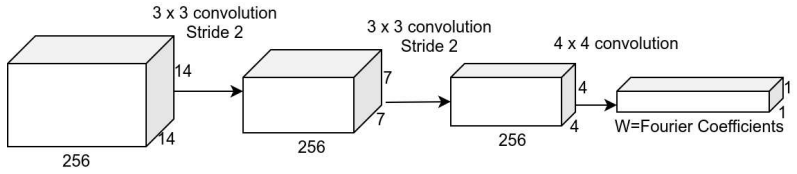


Figure 4.10: Modified FourierMask architecture with spatial size reduced to 1.

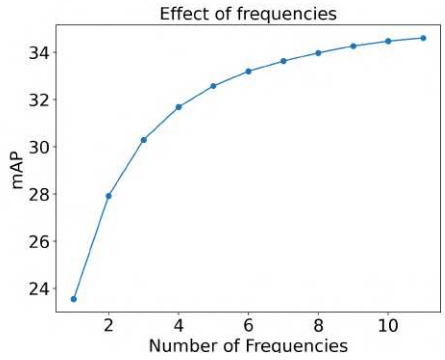
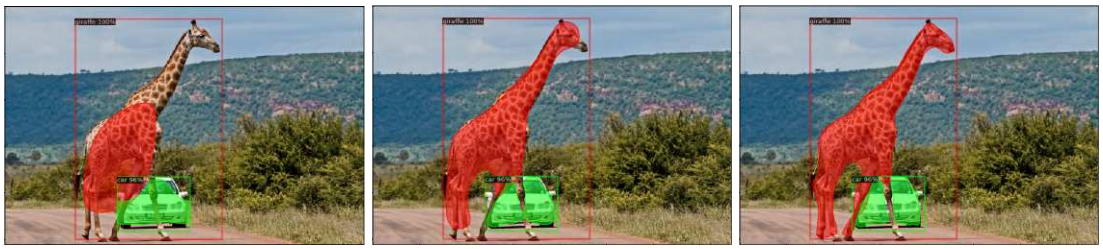


Figure 4.11: The mAP when using a subset of trained frequencies.



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

Figure 4.12: Mask predictions using various frequencies N.

(stride=2) 3×3 convolutions two times (on the ROI) to reduce the feature size by 1/4th and then used a fully connected layer to predict the coefficients. The network architecture is shown in figure 4.10. We applied the equation (4.3) and (4.4) for generating the mask. We copied the predicted Fourier coefficients p times to match the dimensions for matrix multiplication in equation (4.3). We trained the network with 12 frequencies and an output resolution of 56×56 using the IoU loss. We did not add an MLP branch in this experiment and trained only the equation (4.4). We evaluated the mAP precision of the network on the COCO validation dataset when using a subset of the Fourier component frequencies. The network was trained on 12 frequencies, but during inference, we

Model	Backbone	mAP
Mask R-CNN	ResNet-50	34.86
FF	ResNet-50	34.89
FF + MLP	ResNet-50	34.97
FF + MLP	ResNeXt-101	39.09

Table 4.1: Comparison of various FourierMask architectures with Mask R-CNN.

incrementally added the higher frequency components starting from the first component. Figure 4.11 shows the result of this test. The mAP shows a similar trend as seen in figure 4.8 and therefore validates the spectrum analysis and the choice of 12 maximum frequencies. Figure 4.12 shows an example how the masks change when a different number of frequencies are used.

Fourier Features based MLP

To validate that the Fourier Feature-based MLP improves the performance of FourierMask, we trained two networks with the architecture shown in figure 4.7. The network predicts separate Fourier coefficients for each spatial location in this architecture. The first network was trained on the masks obtained using $\mathbf{y}$ in equation (4.4) and output $\mathbf{y}'$ of MLP (FF + MLP). The second network was trained only using equation (4.4) (FF). Both networks used 12 Fourier frequencies and had an output resolution of 28×28 pixels. We used class-agnostic masks and therefore predicted only one class for each region of interest rather than a mask for each class in the COCO dataset. We had two hidden layers (both with sine activations and 256 neurons) and a single output neuron with sigmoid activation. For the first network (FF + MLP), we took the mean of the masks predicted by $\mathbf{y}$ (equation (4.4)) and output $\mathbf{y}'$ of MLP during inference. The results are shown in the table 4.1. As can be seen in the table, the network with an MLP shows the best performance among the models with ResNet-50 backbone. We also trained the same network with a larger ResNeXt-101 (Xie *et al.* (2017)) backbone. The improvement of more than four mAP over Resnet-50 model shows that our model scales well to bigger backbones.

Higher Resolution using Pixel Sub-sampling

One of the advantages of our method is that it can predict masks at sub-pixel resolution because implicit representations are continuous in the input domain. We analyzed this by evaluating the trained networks in section 4.2.3 on the MS COCO validation set on various pixel steps. For the input $\mathbf{x}$ in the equation (4.1), 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 $\mathbf{x}$ by a factor of s. To match the size of input pixels $\mathbf{x}$, we upsampled the coefficients $\mathbf{W}$ in equation (4.3)

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

Table 4.2: Sub-sampling performance and speed (GTX 2080Ti).

in the spatial dimension using bilinear interpolation by a scaling factor 2^{s-1}. Table 4.2 shows the evaluation using the two networks explained in section 4.2.3. We can observe that using a lower pixel step improves the mAP.

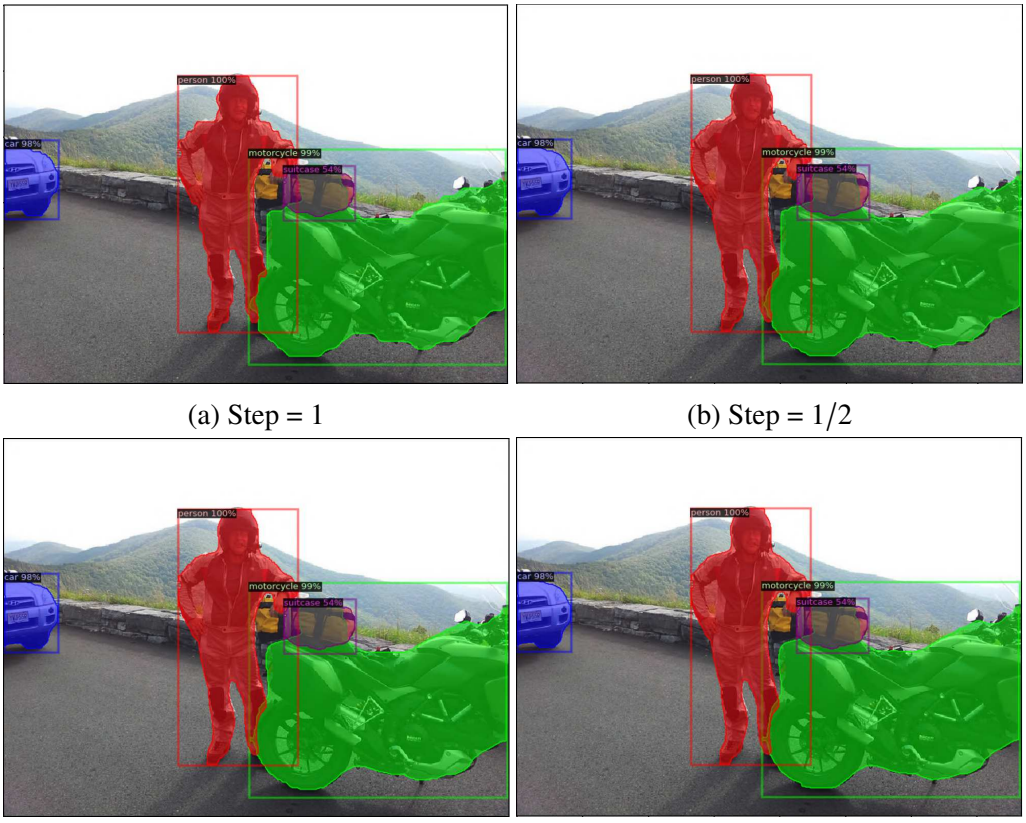
Figure 4.13 shows how the mask boundary smooths out when sub-sampling the pixels. Note that we trained the network on 28×28 output resolution, but we can generate higher resolution output during inference, which is a considerable advantage over other methods.

Higher Resolution using Extended FourierMask

To generate higher resolution masks, we used Extended FourierMask (section 4.2.2) along with the subdivision strategy from Pointrend (Kirillov *et al.* (2020)). We replaced the predictions from equation (4.4) (coarse predictions) with the fine detailed predictions from the extended FourierMask. This resulted in masks which were crisper and boundary aligned. For training extended FourierMask, we employ the default settings of the point selection strategy along with the point loss from PointRend. The results are shown in table 4.3. Here, we also evaluate the mask quality using the *Boundary IOU* (Cheng *et al.* (2021)) metric (mAP_{bound}), which penalizes the boundary quality more than overall correct pixels. Compared to Mask R-CNN, we see a decent improvement of more than 0.7 mAP_{mask} and 1.6 mAP_{bound} with comparable speeds. We can clearly see visual improvements, specially in boundary quality (see figure 4.4). Compared to PointRend, we observe that the masks are more complete (see figure 4.4) with a reasonably lower inference speed. Note that extended FourierMask achieves 224×224 in 3 sub-division steps compared to 5 steps of PointRend because extended FourierMask's initial resolution is 28×28 compared to 7×7 of PointRend.

ReLU vs Sinusoidal Activations

To investigate if sinusoidal activations in MLP indeed perform better than ReLU activations, we trained a network with the same settings and architecture as in the section



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

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

Figure 4.13: Subsampling the pixels smooths out the boundaries of the mask.

Model	Sub. steps	Resolution	mAP_{mask}	mAP_{bound}	Speed (ms)
Mask R-CNN	0	28×28	34.86	21.2	48.7
Extended FourierMask	0	28×28	35.01	21.0	48.7
Extended FourierMask	1	56×56	35.63	22.8	52.4
Extended FourierMask	2	112×112	35.64	22.8	55.7
Extended FourierMask	3	224×224	35.64	22.9	59.4
PointRend	5	224×224	36.12	23.5	81.6

Table 4.3: The effect of subdivision inference.

4.2.3, but replaced sine activations with ReLU. Table 4.4 shows that the usage of sinusoidal activation functions is indeed capable of increasing the performance.

Model	Backbone	mAP
FM + MLP (Sine)	ResNet-50	**34.97**
FM + MLP (ReLU)	ResNet-50	34.41

Table 4.4: Comparison between Sine and ReLU.

4.2.4 Conclusion

In this study we demonstrated how implicit representations combined with the Fourier series can be applied to instance segmentation. Our Fourier mapping generated compact masks. A shape is determined by the lower Fourier frequency and a sharp edge by the higher Fourier frequency. Furthermore, by subsampling 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. The boundary quality of FourierMask is significantly improved when using our extended FourierMask associated with a renderer MLP.

Qualitative comparisons

In the following are some sample images from extended FourierMask compared to Mask R-CNN with similar architecture. The left images are the prediction from extended FourierMask with subdivision inference and the right ones are of Mask R-CNN. Since our method improves the boundary quality, we choose large objects with many corners/edges (for example giraffes, humans, airplanes, vehicles and other animals) rather than smooth objects.

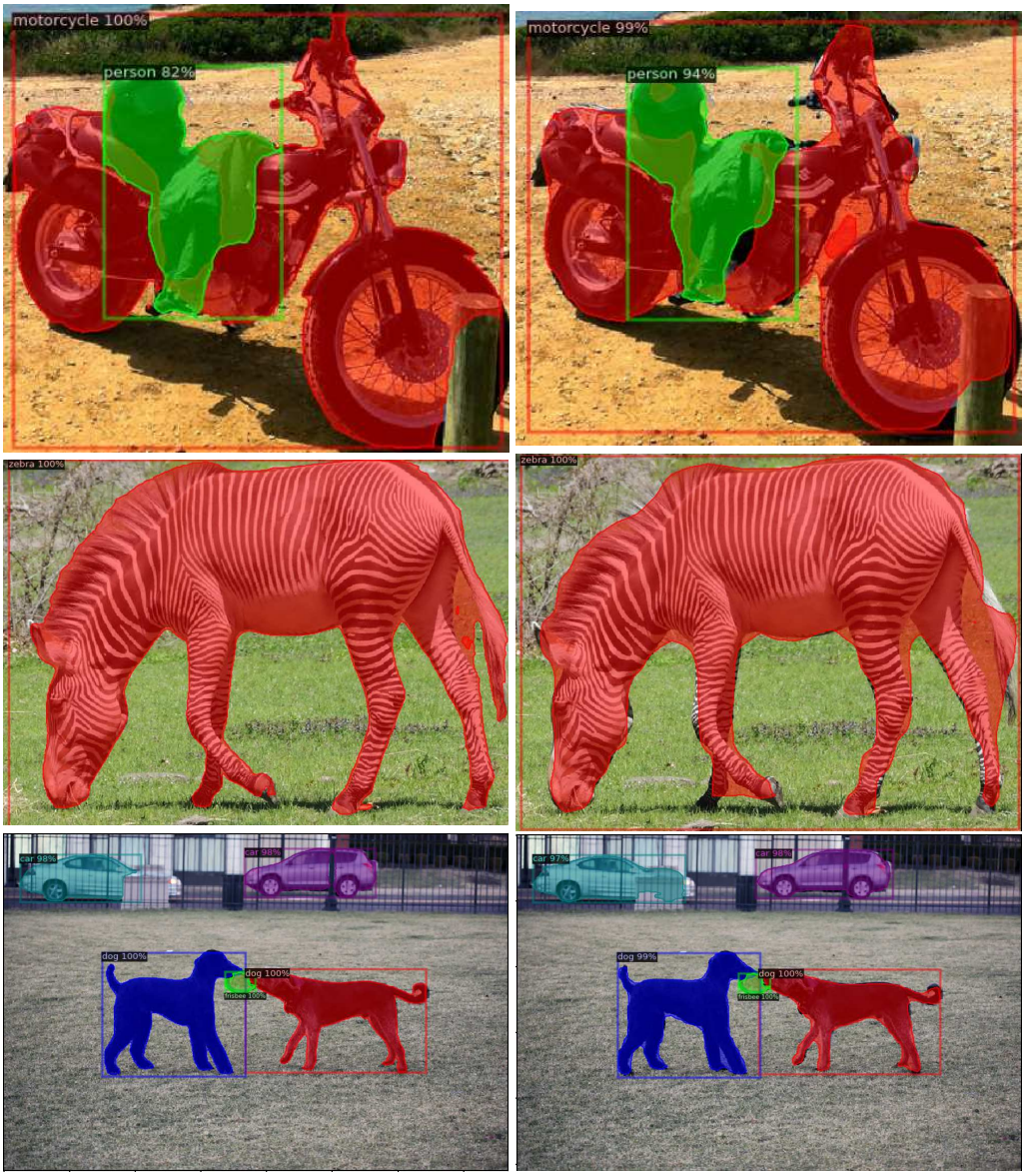


Figure 4.14: Left image: FourierMask, Right image: Mask R-CNN.

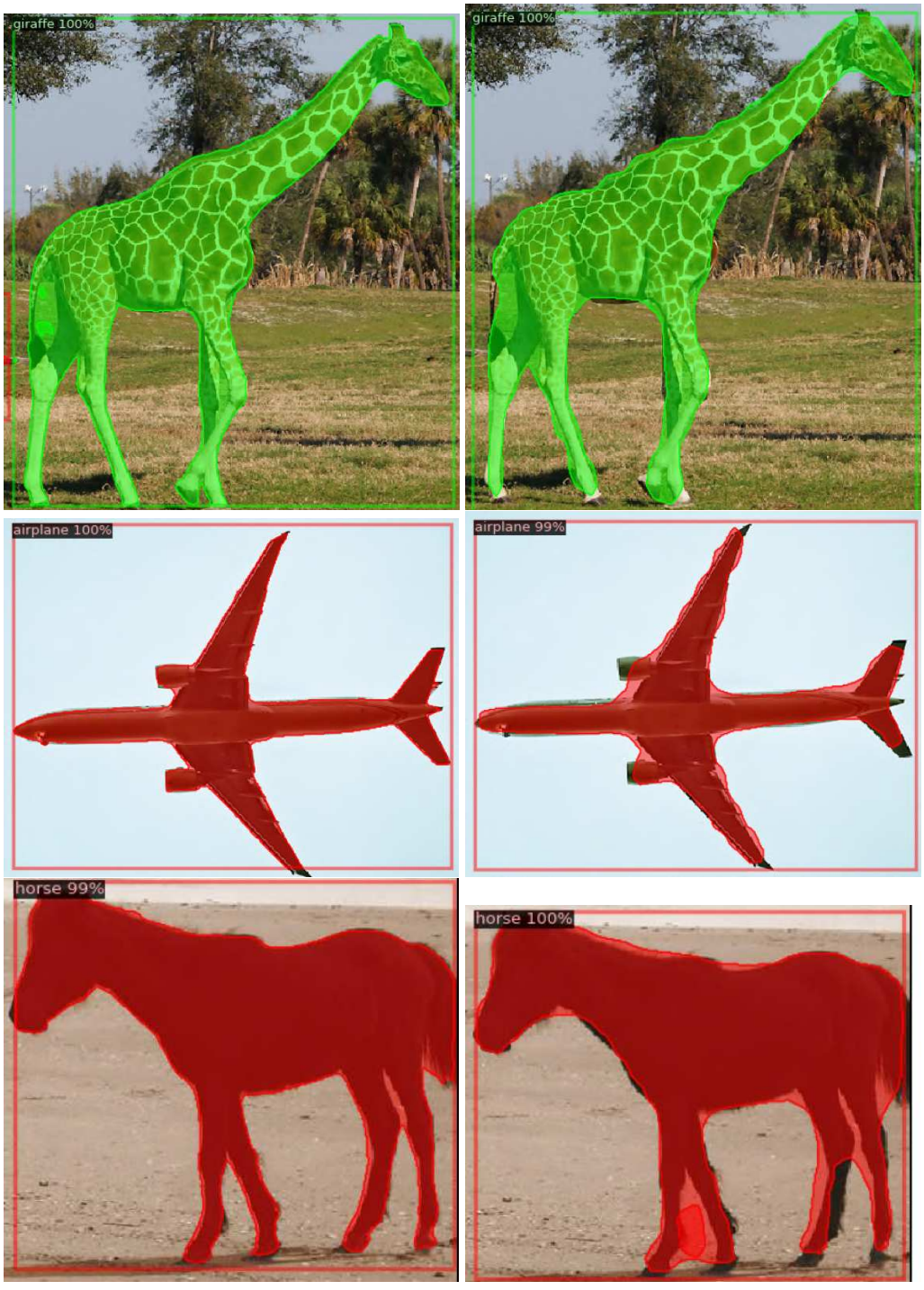


Figure 4.15: Left image: FourierMask, Right image: Mask R-CNN.

Chapter 5

Pose Estimation with Augmented Autoencoders

Object Pose A full object pose is defined as the three-dimensional translation (x,y,z position) and three-dimensional orientation (in any parameterization) of an object. As discussed in section 2.1.1, the three degrees of freedom in a rotation refer to the three freely eligible elements. This problem has been studied in a variety of ways. 2D object detection algorithms are limited to the two translation parameters in the image plane and ignore the third translation parameter and the object's orientation. Using depth estimation methods, one can obtain a third out-of-plane translation parameter. As we will discuss below, some recent works have only focused on the 3D orientation of the 6D problem, while others focus on the full 6D problem. Typical challenges associated with pose estimation include:

- Occlusion: Occlusion occurs when part of the object is hidden by another object, making it difficult for the pose estimation system to accurately predict its position and orientation.
- Scale and viewpoint changes: Pose estimation algorithms often rely on certain assumptions about the size and viewpoint of the object being tracked. If the object changes size or the viewpoint changes, the algorithm may not be able to accurately estimate its pose.
- Limited training data: Many pose estimation algorithms are based on machine learning techniques, which require a large amount of training data to learn from. If the training data is limited or not representative of the real-world conditions, the algorithm may not be able to accurately estimate the pose of the object.
- Real-time performance: Pose estimation algorithms often need to run in real-time, which can be challenging due to the computational complexity of the algorithms and the need to process large amounts of data in a short amount of time.
- Ambiguity: Some objects, such as a cylinder or cube, have a high degree of symmetry and may have multiple possible poses that are equally valid. This can make

it difficult for pose estimation algorithms to determine the correct pose of the object.

5.1 Solutions to Pose Estimation

Pose estimation for 6D objects was almost considered solved at one point. Unfortunately, the detection systems were only capable of detecting objects with rich texture. The research focused on finding features that improved the robustness of the system to changes in illumination. As time went on, more methods were developed that focus on objects with little or no texture. With applications in robotics becoming more intriguing, which required algorithms that were robust to occlusions and clutter in the scene, new datasets illustrating these issues were developed. In this regard, the BOP challenge has to be named. It took place for the first time in 2019 and from then on annually see Hodaň *et al.* (2020). It is a challenge for researchers on 6D pose estimation on multiple well-known 6D datasets such as YCB in Xiang *et al.* (2017), containing textureless and symmetric objects, strong occlusions and clutter. The Linemod dataset in Hinterstoisser *et al.* (2012) also contains symmetric objects and even has a subset that is named occluded Linemod, which follows its name containing heavily occluded scenes of objects. The last we want to name here is TLESS (Hodan *et al.* (2017)), which contains textureless industrial objects that form a challenging benchmark for pose estimation. Recently, convolutional neural networks (Sundermeyer *et al.* (2020); Labbé *et al.* (2020); Park *et al.* (2019b); Li *et al.* (2019)) have proven to be promising in the BOP2020 challenge (Hodaň *et al.* (2020)) on 6D pose estimation, even surpassing the top depth-based methods for the first time. They also tend to be faster, mostly with a runtime of less than one second. These methods mainly depend on an object detection phase, which is achieved using a state-of-the-art object detector (Mask R-CNN He *et al.* (2017), RetinaNet Lin *et al.* (2017a), Faster R-CNN Ren *et al.* (2015)).
We are going to present some works sorted by their technique in the following.

Traditional Template-based and Feature-based Methods

In the past, most techniques for 6D pose estimation were feature-based or template-based. Traditional approaches include template-matching (e.g. the work of Hinterstoisser *et al.* (2011)), where they try to match a rotating object to the image until a good fit is achieved. However, these approaches suffer especially from occluded scenes. The feature-based algorithms locate visual characteristics that correlate to known object positions and then calculate the pose. One differentiates between local feature matching, which includes the finding of point-to-point correspondences of multiple images such as Multiview Stereo by Schönberger *et al.* (2016) and Goesele *et al.* (2007) and structure motion by Mur-Artal *et al.* (2015). Here, key points are detected for input images, such as corners or edges. Then point-to-point correspondences are found by matching simi-

lar features in the different images. In contrast to the multiview approaches, where the movement of the camera is estimated, key points can directly be used for pose estimation which has been done for point clouds by Tombari *et al.* (2010) and Rusu *et al.* (2009) and RGB images in Lowe (2004) and Bay *et al.* (2006). Methods based on local features suffer from textureless objects when RGB data is used as input and symmetric objects for the point cloud-based methods. To achieve better results for textureless objects, methods relying on global features have been introduced. The advantage of global features is that they inherit representations of objects in different poses and thus do not suffer from the non-uniqueness of local features. Global-feature-based methods have similarly been used for different input data such as RGB by Konishi *et al.* (2016), RGB-D by Hinterstoisser *et al.* (2012); Tejani *et al.* (2014) and point cloud by Drost *et al.* (2010); Wohlkinger and Vincze (2011). With the presence of a 3D CAD model, dominantly point pair features have been used, such as by Drost *et al.* (2010).

Deep Learning-based Methods

With the appearance of convolutional neural networks, researchers made the previously hand-designed features learnable. In the work of Kehl *et al.* (2016) they make use of convolutional autoencoders to train local descriptors on RGB-D data. Each local correspondences of objects vote for the center based on the relative position of the patch. Similarly, global features can be encoded by convolutional neural networks. In the works by Wohlhart and Lepetit (2015); Balntas *et al.* (2017) global features are trained by learning a manifold that distinguishes between different poses in the feature space.
Other methods directly regress the pose. This can be achieved by extending Mask R-CNN to pose estimation by adding a pose branch to the network output, see Do *et al.* (2018). In their work, they regress the SO(3) Lie group as the pose representation; they motivate this representation by saying that compared to quaternions and orthonormal matrix, the Lie algebra has fewer parameters and is unconstrained and thus making the training easier. In general, different types of rotation parameterizations have been used for the regression task. For example, the 2D projections of the 3D bounding box coordinates by Rad and Lepetit (2017), or unit quaternions and translations (Xiang *et al.* (2017)). Overall, the direct regression is computationally efficient as there is no need to further process the network output. These methods, however, have the drawback of not generating multiple pose hypotheses to estimate occluded objects robustly, and symmetric objects have to be handled accordingly.

Implicit and Statistical Pose Estimation Methods

As symmetries occur plentifully in industrial or everyday objects, it is interesting and essential to conduct further research on their occurrence. If object symmetries are known during training, it is possible to group equivalent rotations to a single one, allowing training to proceed as in classical single-valued regression (Pitteri *et al.* (2019)). In

Corona *et al.* (2018), manually labelled symmetries of 3D poses are needed to learn the embedding and classification of the symmetry order together.

On the contrary, Sundermeyer *et al.* (2020) make pose or symmetry supervision unnecessary by using an implicit augmented autoencoder to isolate pose information. During inference, they receive a latent representation, compare it to a fully covered sample in a codebook of saved latent representations of rotations and take the closest one.

As symmetries are not the only source of pose uncertainty, it is interesting to utilize a more flexible representation. Recent works focused on a statistical approach by considering parametric probability distributions. Peretroukhin *et al.* (2020); Deng *et al.* (2022); Gilitschenski *et al.* (2019) regressed the parameters of a von Mises distribution over Euler-angles and Mohlin *et al.* (2020) utilize Matrix Fisher distributions on SO(3). To this end, Prokudin *et al.* (2018), Gilitschenski *et al.* (2019) and Deng *et al.* (2022) propose using multimodal mixture distributions. In Deng *et al.* (2022) they introduce Deep Bingham networks, a framework to handle pose ambiguities. They introduce a multi-hypothesis head to predict a family of poses to capture the nature of the solution space. From a technical perspective, they regress Bingham mixture models. Here, the Bingham distribution lies on $\mathbb{S}^{d-1}$ and is an antipodally symmetric probability distribution derived from a Gaussian with zero mean. While in Gilitschenski *et al.* (2019), they try to deal with the uncertainty of orientation, they introduce a loss to capture the symmetries by characterizing uncertainty with unit quaternions based on the Bingham distribution. They name their introduced loss 'Bingham loss'. Furthermore, they demonstrate multi-modal orientation prediction by using a Bingham variant of mixture density networks. In Prokudin *et al.* (2018) they propose a probabilistic deep learning model to predict a mixture of von Mises distributions. With that, they are able to learn a mixture model using a finite and infinite number of mixture components. Furthermore, they give an analysis of the importance of probabilistic regression. One challenge when training the mixtures is avoiding mode collapse, for which a winner-takes-it-all strategy can be used (Deng *et al.* (2022)). An alternative to the mixture models is to predict multiple pose hypotheses directly Manhardt *et al.* (2019), but this does not share any of the benefits of a probabilistic representation.

A more general representation of the distribution is proposed by Murphy *et al.* (2021), where they implicitly model the probability density function with a multilayer perceptron whose architecture is inspired by the field of INRs. Their works provide the challenging SYMSOL and SYMSOL II datasets focused on symmetries and can show superior performance to the above-introduced mixture models.

5.2 Object Detection and Autoencoder-based 6D Pose Estimation for Highly Cluttered Bin Picking

In this section, we are going to introduce our work in Höfer *et al.* (2021), a framework for object detection and pose estimation developed for the special setting of highly cluttered bin picking. Bin picking is a fundamental problem in industrial contexts and robotics, with 6D pose estimation serving as its primary module. However, when it comes to small objects, industrial depth sensors are inaccurate. As a result, we offer a framework for implicit pose estimation in extremely cluttered situations with small objects that relies mostly on RGB data and only uses depth information for pose refining. We compare synthetic data generation methodologies for object detection and pose estimation in this study, and we present a pose filtering system that finds the most accurate predicted poses. By the usage of synthetic data, we overcome the problem of limited training data, and as it is the case that industrial objects do not change in size, such as humans do, we do not suffer from the problem of scale changes. As we assume our system is with a fixed camera position, we generate the synthetic data accordingly from the same viewing angle as it is given in the real scenario, removing the viewpoint change problem. Furthermore, we show that our pose estimation algorithm can process multiple images per second, which is considerably faster than traditional methods, e.g. relying on template matching. Finally, by the implicit nature of the augmented autoencoder, no symmetry supervision is necessary, helping in the presence of symmetric objects.

5.2.1 Introduction

Bin picking is a major automation task with various applications in industrial sectors. The core starting problem of this work is the 6D pose estimation of instances. To tackle this problem, an RGB-D or depth camera is usually installed on top of the bin. There are existing solutions to bin picking of large objects, mostly using local invariant features (Abbeloos and Goedemé (2016); Liu *et al.* (2018)) or template-matching algorithms (Hinterstoisser *et al.* (2011)), which rely on the computationally expensive evaluation of many pose hypotheses. Moreover, local features do not perform well for texture-less objects, and thus, template-matching often fails in heavily cluttered scenes with severely occluded objects. Additionally, depth sensors are often more sensitive to lighting variations than RGB cameras (Sundermeyer *et al.* (2020)). Most importantly, for small objects, the depth information is often insufficient to get accurate pose estimates. Therefore, in this work, we focus on RGB-based convolutional neural networks, which make use of depth information only for pose refinement.

Accordingly, one of the important issues for training a deep network is labelling the training dataset, which requires high effort for tasks like 6D pose estimation (Hodan *et al.* (2017)). Given a CAD model of the object, which is usually available in the industry, generating a synthetic dataset is possible. However, training on only synthetic 2D

images of the CAD models does not generalize well to real data. Hence, more insightful techniques are required to bridge the gap between simulation and reality (Sundermeyer *et al.* (2020)).

Generally, a state-of-the-art object detector is first used to recognize individual objects, and the resultant cropped images are passed to the pose estimator. Following Labbé *et al.* (2020); Joffe *et al.* (2019), we use Mask R-CNN (He *et al.* (2017)) for object detection. As for the task of pose estimation, we consider an implicit augmented autoencoder (Sundermeyer *et al.* (2020)) since it has demonstrated good performance in bin picking of deformable products (Joffe *et al.* (2019)). Sample results of our proposed method are displayed in Fig. 5.10. Moreover, once the general pipeline is given, the predicted poses can be further refined using a pose refinement method. Previously, this step has been achieved (Labbé *et al.* (2020); Sundermeyer *et al.* (2020)) by the ICP algorithm (Zhang (1994)). As the ICP-based methods show slow performance, we show that incorporating the depth information into the pose estimation procedure achieves comparable results. Besides, a filtering algorithm is applied to choose the best poses among the estimated ones. The main contributions of this work are as follows:

1. We present a comprehensive framework from creating a synthetic dataset to the prediction of the 6D pose estimates in bin picking scenarios, where no real labelling is needed.

2. We show that a more realistic renderer for data generation significantly improves the performance on heavily cluttered piles.

3. We present a pose filtering scheme to select the best pose predictions.

4. We give an analysis of how the performance of the autoencoder can be improved in bin picking scenarios.

5.2.2 Methodology

In this work, we consider heavily cluttered and occluded scenes of small industrial objects. Here, we explain the methods for dataset generation, followed by the full framework for object detection and pose estimation.

Dataset Generation

Creating a synthetic dataset can be achieved by using the CAD models of the objects. Since our pipeline has two main tasks, we need to create a dataset for both, object detection and pose estimation. As the effort of labeling 6D poses in cluttered scenes is high and demands a complex setup (Hodan *et al.* (2017)), some works (Kehl *et al.* (2017); Sundermeyer *et al.* (2020)) have proposed training on synthetic images rendered from

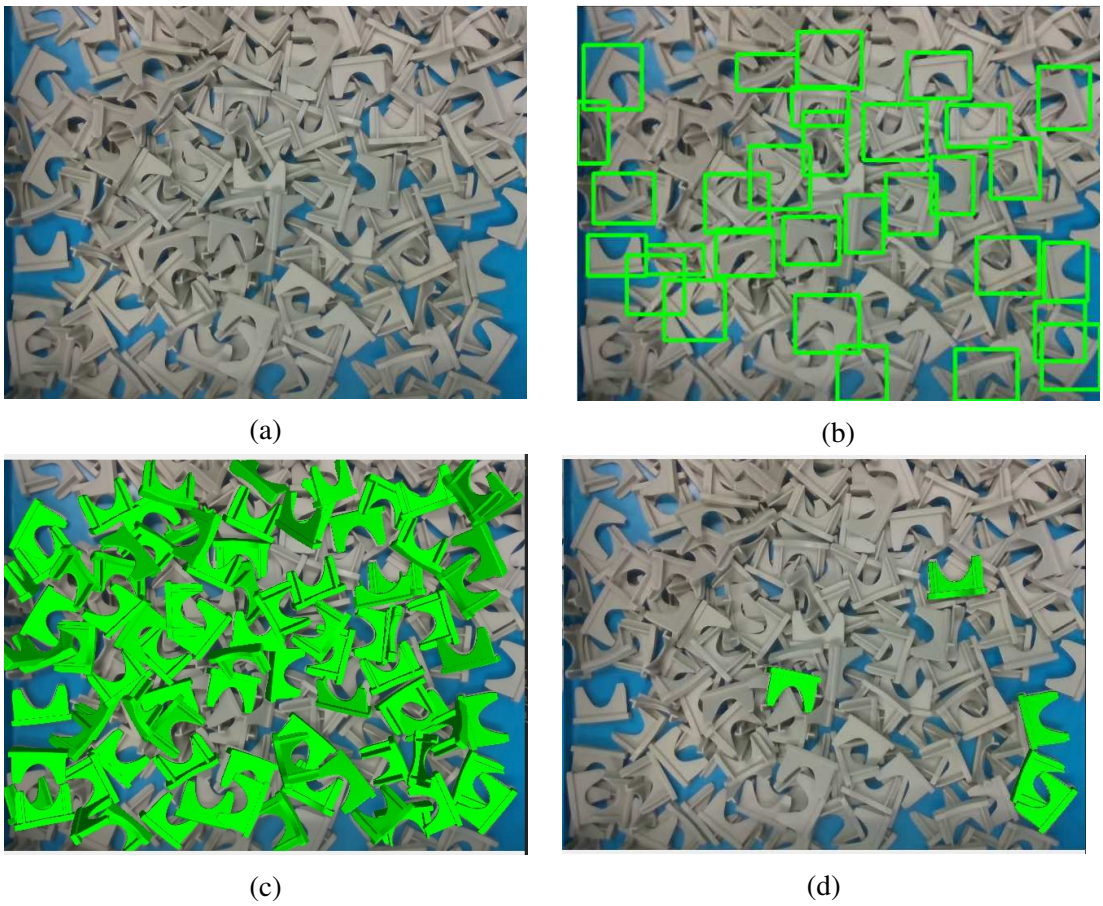
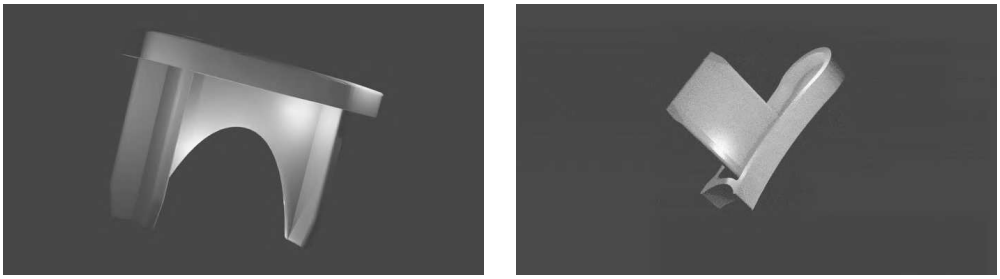


Figure 5.1: (a) A sample image of a cluttered bin captured by a Microsoft Azure Kinect RGB-D camera. (b) Detection results, limited to 30 objects, to be visually recognizable. (c) Pose estimation results for all the detected objects. (d) The best five selected poses based on the filtering algorithm.

a 3D model. To bridge the gap to reality, random augmentations and domain randomization techniques have been applied (Pashevich *et al.* (2019); Tobin *et al.* (2017)). As a different solution to this problem, a more realistic data generation using a physics engine has been proposed in Denninger *et al.* (2019). In our work, we benefit from this approach to generate photorealistic cluttered piles and propose the full framework for object detection and pose estimation.

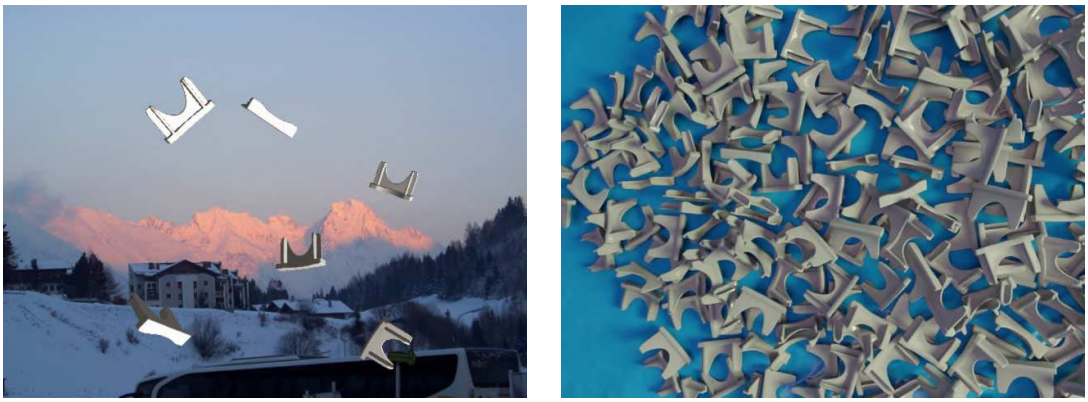
Dataset for Object Detection

As the first approach, we make use of the pipeline in Sundermeyer *et al.* (2020) to generate synthetic images to train the object detector. In particular, a CAD model is used to render the object on a black background. Then, random images from the Pascal VOC dataset Everingham *et al.* (2015) are added as a background, followed by random aug-



(a) Object 1 (b) Object 2

Figure 5.2: The objects of interest are grey plastic pieces with sizes of $2.3 \times 3.6 \times 0.8\text{cm}^3$ and $1.5 \times 2.7 \times 0.9\text{cm}^3$, respectively.



(a) Naive dataset (b) Realistic dataset

Figure 5.3: The synthetic datasets generated with two different pipelines for object 1.

mentations strategies, including blurring, contrast normalization, dropout, cropping, inverting and more. We create 60K images per object with 5-20 instances per scene. Due to their simplicity, we call these images "naive dataset".

For the second approach, we employ BlenderProc (Denninger *et al.* (2019)) to generate more realistic synthetic images. BlenderProc utilizes a physics engine to make the synthetic data look more realistic. Furthermore, it uses different lighting effects, object materials and applies physics and collision checking as well. With BlenderProc, we generate for each object type 20K images with 30 instances and 5K images with 300 objects. The camera configuration is sampled in a range of 20° around the top of the scene with a height between 27-33cm. We call this the "realistic dataset".

In this work, we consider two industrial objects (see Fig. 5.2). Sample images of the naive and realistic images for object 1 are depicted in Fig. 5.3.

Dataset for Pose Estimation

Similar to the naive dataset in the previous section, we generate images with corresponding 6D pose annotations, with an additional step of image cropping around the objects. We also compare the original pipeline results against a new dataset, where we render multiple objects in the image crops. In Fig. 5.6, samples of different training data are displayed on the left side. While the top image shows one single object in each image crop, the bottom one includes multiple objects.

Test Dataset for Object Detection

To evaluate the object detector, we captured 50 real images per object model with more than 100 instances in the bin. The images were taken using a Microsoft Azure Kinect camera mounted at a height of 30cm over the bin (see Fig. 5.4). We hand-labelled the bounding boxes and segmentation masks for all the present objects.

Test Dataset for Pose Estimation

While we show qualitative results in real-world scenarios, the quantitative results are reported on a synthetic dataset because only for this we have full ground truth data. The autoencoder is trained on synthetic data following the pipeline in Sundermeyer *et al.* (2020), and we create the test dataset with BlenderProc (Denninger *et al.* (2019)). To be more precise, BlenderProc generates 3D scenes, whereas the pipeline in (Sundermeyer *et al.* (2020)) creates augmented 2D images. Since the data distributions of these synthetic datasets are different, we will show that training the pose estimator on the naive dataset and testing on the photorealistic images leads to suitable performance.

As such, for each object, we created one dataset consisting of 1K images with 300 objects. The camera is located 30cm on top of the bin ground. We choose a blue background to make it visually comparable to our real settings.

Object Detection

In general, any state-of-the-art object detector (Faster R-CNN (Ren *et al.* (2015)), RetinaNet (Lin *et al.* (2017a)), SSD (Liu *et al.* (2016))) can be used for object detection. However, these methods only predict the bounding boxes. Therefore, we choose Mask R-CNN, which has the advantage of predicting the segmentation masks of the object as well. This can be used for pose refinement (Wong *et al.* (2017)) by segmenting the point clouds of the objects. In addition, we can compare the pose estimation results when, instead of the whole bounding box, only the pixels visible in the segmentation mask are given to the pose estimation module. Given an image, we predict a set D of object detections.

(a) Real test dataset 1 (b) Real test dataset 2

Figure 5.4: Examples of our labeled test dataset in real scenarios.

Pose Estimation

To receive pose estimates from the set D of the detections, we resort to the autoencoder network presented in Sundermeyer *et al.* (2020). As the autoencoder's training procedure is based on only synthetic data, it is more applicable to new industrial settings where no labelled data exists. In addition, the autoencoder has demonstrated good performance in pick-and-place tasks (Deng *et al.* (2020b); Joffe *et al.* (2019)). Its method of operation is as follows: An autoencoder is originally a dimensionality-reducing technique for high dimensional data such as images, consisting of an Encoder ϕ and a Decoder ψ (see Fig. 5.6). An input $x \in \mathbb{R}^O$ will be first fed through the Encoder: $z = \phi(x)$. This value $z \in \mathbb{R}^o$ will later be called the latent representation of the vector x. For the dimension we have $o << O$. The output is produced by the Decoder: $\hat{x} = \psi(z)$. Putting it all together, we have:

$$\hat{x} = (\psi \circ \phi)(x)$$

In our case, we will have the same dimensions for the input x and the output $\hat{x}$.
In the training procedure, we first produce a rendered image of the object with some random rotation on black background; we call it I_1. A random background is added, and further augmentations are done on this image. Let us say f_{augm} is the function responsible for these operations. The new image $I_2 = f_{\text{augm}}(I_1)$ is then fed as the input to the Autoencoder. The training objective is to reconstruct the input I_1 after passing through the low dimensional bottleneck, where the loss is the sum over the pixel-wise L2 distance. Here ϕ and ψ are neural networks that are being trained.
After training, we create a codebook to determine the rotation of the object. The code-

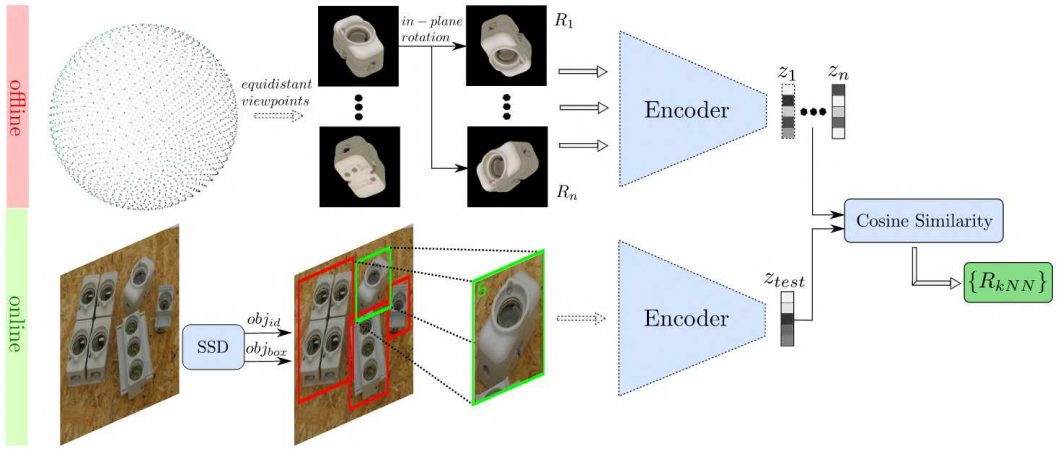


Figure 5.5: The Autoencoder way to do pose estimation. After training the Autoencoder there is an additional offline stage to create the codebook, which is visible at the top. Equidistant viewpoints are sampled (typically a 5° difference). The object of interest is then rendered on a black background without further augmentations. Now, each of these rendered poses is fed through the encoder to generate multiple latent representations $(z_1, \ldots, z_n)$. This so-generated matrix is now saved with the corresponding rotations and referred to as the codebook. During the online stage displayed at the bottom, an object detector (here SSD) is used to detect an object and produce a bounding box. The image within the bounding box is then cropped and used as the input to the encoder of the Autoencoder. Again a latent representation is produced, and a knn-search finds then the most similar representation in the codebook via cosine similarity. Credit: Sundermeyer *et al.* (2020).

book is the set of all latent representations of the discretized 3D rotations that cover the whole SO(3).
At test time, the image crops from the set D are fed into the encoder. The resulting latent representation z_{test} is then compared with all the latent representations (z_i) from the codebook via a k-NN search, with the similarity function as:

$$cos_i = \frac{z_i z_{\text{test}}}{||z_i|| \, ||z_{\text{test}}||}. \tag{5.1}$$

We then choose the rotations with the highest cosine similarities. For a detailed illustration see figure 5.6.

5.2.3 Selecting the Best Pose Estimates

In cluttered scenes, we have pose estimations of several hundred objects. These will be used for the picking task, and as the robot will only pick one object at a time, we are

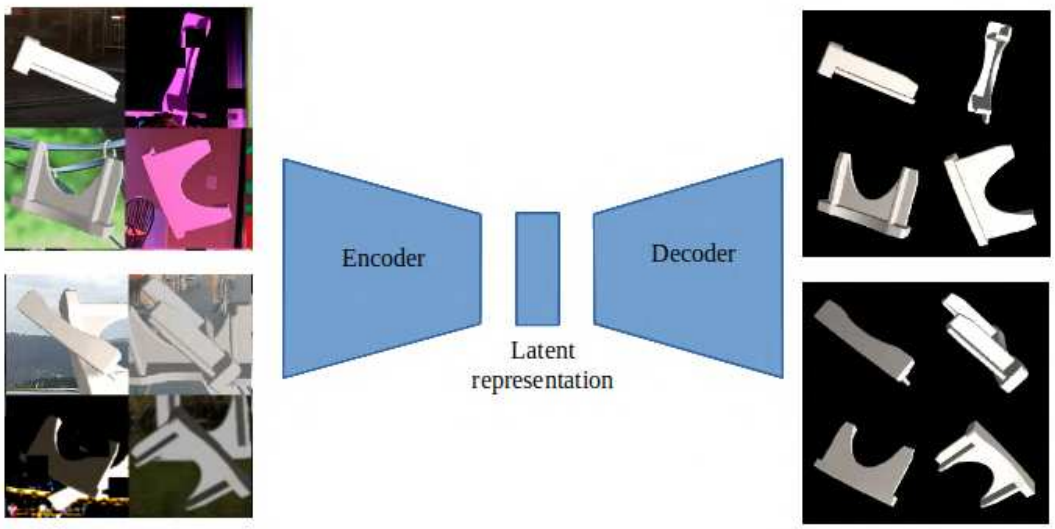


Figure 5.6: The architecture of the autoencoder. The autoencoder is trained to map the augmented images to the original image. On the left, there are the two different types of training data.

only interested in the k-top pose estimates, and the question arises, how to select the k-best pose estimates. While one can sort the pose estimates regarding to the scores given by Mask R-CNN, or by the highest cosine similarities (5.1), we define a new selecting method that compares the depth of the original image with the depth of the rendered image.
Let A_1^i be the predicted segmentation mask of object i.
Given a predicted 6D pose $(\bar{t},\bar{R})$, we render the depth image $\bar{\Omega}$ and we define the set of pixels $A_2^i := \{(p,q) : |\Omega(p,q) - \bar{\Omega}(p,q)| < m\}$, for some margin m and where Ω represents the real depth image.
A_3^i is the segmentation mask of the rendered image.
$A^i := A_1^i \cap A_2^i \cap A_3^i$. For each pose estimate we calculate the depth error as follows:

$$e_i = \sum_{(p,q)\in A_i} |\Omega(p,q) - \bar{\Omega}(p,q)| \tag{5.2}$$

In the next section, we compare the different approaches.

5.2.4 Experimental Results

Experiments on Object Detection

We fine-tuned a pretrained Mask R-CNN with a ResNet-50 backbone for 15 epochs with an initial learning rate of 0.001 and a mini-batch size of 4 images. The learning rate was

Object	dataset	$AP_{50:95(\%)}$	$AP_{50(\%)}$	$AR^{max=100(\%)}$
Object 1	naive	9.3	12.5	9.6
	realistic	**66.8**	**82.8**	**80.3**
Object 2	naive	0.9	1.0	0.1
	realistic	**50.4**	**68.7**	**59.2**

Table 5.1: Object detection results after training Mask R-CNN on different synthetic datasets

reduced by a factor of 10 at epochs 3, 6, 9 and 12. Stochastic gradient descent (SGD) with momentum (0.9) and weight decay (0.0005) was used for optimization. Our work is based on the torchvision implementation of Mask R-CNN that can be found in (Marcel and Rodriguez (2010)). In Table. 5.1, the accuracies of object detection in terms of AP_{50} (the average precision with IoU thresholded at 0.50), $AP_{50:95}$ and $AR^{max=100}$ (the average recall with 100 detections per image) are tabulated. While training on the naive dataset does not generalize well to our heavily cluttered real scenarios, Mask R-CNN trained on the realistic dataset considerably boosts the performance.

Experiments on Pose Estimation

To this goal, we trained the autoencoder with a latent space size of 128. We chose the L2 loss function, a learning rate of 0.0001 and used the Adam optimizer with a batch size of 32 and trained it for 40K iterations. The pose error metrics used for evaluation are the Visible Surface Discrepancy (VSD), the Maximum Symmetry-aware Surface Distance (MSSD) and the Maximum Symmetry-aware Projection Distance (MSPD), that are being used in the BOP2020 challenge (Hodaň *et al.* (2020)). An estimated pose is considered as correct w.r.t. the pose-error function e if $e < \theta_e$, where $e \in \{e_{\text{VSD}}, e_{\text{MSSD}}, e_{\text{MSPD}}\}$ and θ_e is the threshold of correctness. We used the same values for θ_e as in Hodaň *et al.* (2020) to calculate the average recall rates AR_{VSD} , AR_{MSSD} and AR_{MSPD}. The performance of the method on a dataset is measured by the Average Recall $AR = \big(AR_{\text{VSD}} + AR_{\text{MSSD}} + AR_{\text{MSPD}}\big)/3$.

In Table 5.2, we compare the results of the different methods on selecting the best five pose estimates. In these experiments, we did not make use of ICP, but we took the depth measurement at the center of the object. It shows that sorting according to the vector (e_i) in (5.2), results in superior performance compared to other approaches.

The experiments in Table 5.3 are conducted using the selection method defined by the vector (e_i). We compare the results, when testing with only RGB information, using the depth measurement at the object center and the improvement through ICP refinement. While ICP refinement increases the performance slightly, the refinement of several hundred poses per image takes time, making it impractical for the usage in real-time settings. The experiments to reduce the noise in cluttered scenes, like feeding only the pixels vis-

Object	sorted by Mask R-CNN scores	sorted by max cosine sim.	sorted by depth differences (e_i)
Object 1	0.509	0.394	**0.812**
Object 2	0.533	0.449	**0.633**

Table 5.2: Average recall of top 5 pose estimates sorted by three different approaches for selecting the best estimates

Object	RGB	RGB + depth			
		+ ICP	normal	mult.-obj.	mask
Object 1	0.691	**0.829**	0.812	0.790	0.798
Object 2	0.348	**0.703**	0.633	0.627	0.614
time (s)	**0.699**	18.39		**0.697**	

Table 5.3: Average recall of top 5 pose estimates using ICP and depth measurements and combinations.

ible in the mask to the autoencoder, or training the autoencoder with multiple objects, have shown, against our intuition, a worse performance than the normal pipeline. Note how incorporating the depth has improved the accuracy.

5.2.5 Conclusion

In this work, we investigated the task of bin picking from piles of crowded, small-sized and identical objects. In particular, we explored the main required vision modules for this challenging problem, i.e. object detection and pose estimation. For each task, we employed convolutional neural networks, which are trained on two types of generated synthetic datasets. Experimental results on synthetic and real images show that the proposed comprehensive framework, from dataset generation to pose estimation, is promising for industrial bin picking.

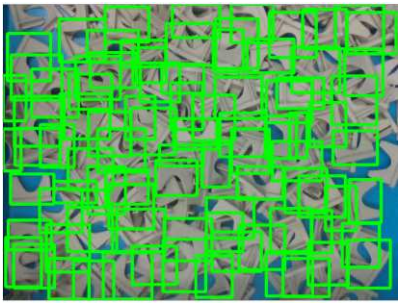
5.2.6 Further Qualitative Results

We provide qualitative results on real scenes captured with a Microsoft Kinect Azure Camera. The images were taken at a height of 30cm above the bin. We visualize all the steps from detection to pose selection. The images were taken on different day times. Therefore the different light conditions affect the colour of the images.

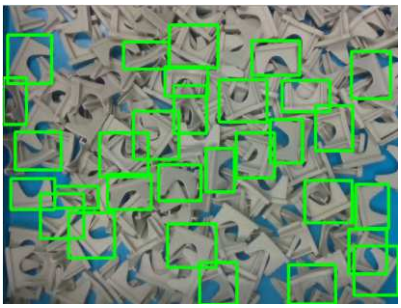
While Mask R-CNN, trained on the naive dataset, detects some objects in a less cluttered scene, its performance on a cluttered bin is not sufficient. On the other hand, the performance after training on the realistic synthetic data yields promising results even on the cluttered scenes.

The pose estimates in the less cluttered scenes look overall good, whereas, in the cluttered scenes, some are incorrect. However, we can see that all selection methods are able to choose visually correct appearing poses. If we take a closer look, the pose estimates selected according to the cosine similarity values in the images 5.7h and 5.9h have both an incorrect pose for the respective upper objects. Also, for the image 5.9g, the lowest pose estimate has a different angle than the original object. In these cases, the selection according to the vector (e_i), could determine those poses as incorrect and they were therefore not selected.

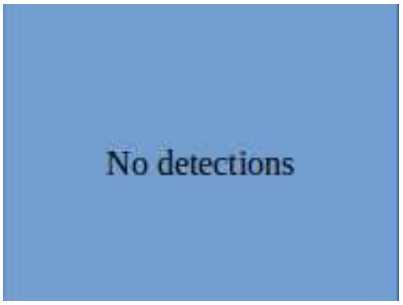
(a) A sample image of a cluttered bin captured by a Microsoft Azure Kinect RGB-D camera.



(b) Detection results of Mask R-CNN trained on the realistic synthetic data.



(c) Detection results of Mask R-CNN trained on the realistic synthetic data, limited to 30 detections.



(d) Detection results of Mask R-CNN trained on the naive synthetic data - no detected objects.

(e) Pose estimation results for all the detected objects.

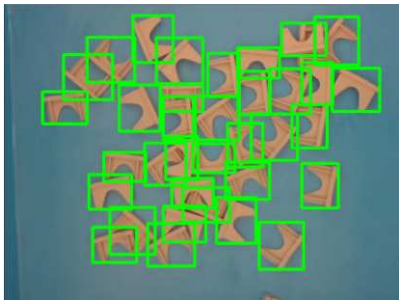
(f) Selecting the best 5 poses according to the vector (e_i).

(g) Selecting the best 5 pose estimates according to the scores of Mask R-CNN.

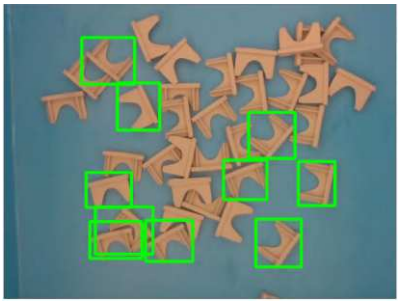
(h) Selecting the best 5 pose estimates according to the cosine similarity values.

Figure 5.7: Qualitative results in a heavily cluttered bin for object 1.

(a) A sample image of a less cluttered bin captured by a Microsoft Azure Kinect RGB-D camera.



(b) Detection results of Mask R-CNN trained on the realistic synthetic data.



(c) Detection results of Mask R-CNN trained on the naive synthetic data.

(d) Pose estimation results for all the detected objects.

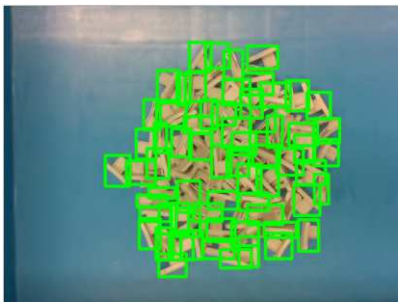
(e) Selecting the best 5 poses according to the vector (e_i).

(f) Selecting the best 5 pose estimates according to the scores of Mask R-CNN.

(g) Selecting the best 5 pose estimates according to the cosine similarity values.

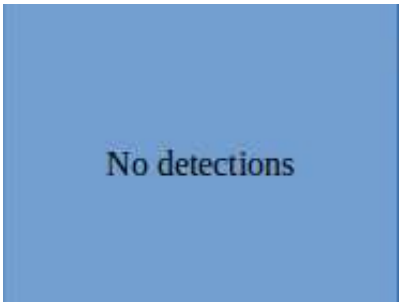
Figure 5.8: Qualitative results in a less cluttered bin for object 1.

(a) A sample image of a cluttered bin captured by a Microsoft Azure Kinect RGB-D camera.

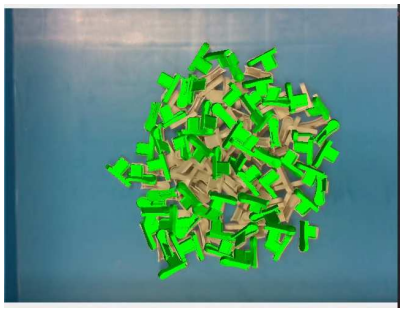


(b) Detection results of Mask R-CNN trained on the realistic synthetic data.

(c) Detection results of Mask R-CNN trained on the realistic synthetic data, limited to 30 detections.



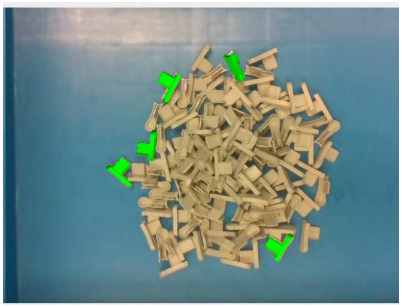
(d) Detection results of Mask R-CNN trained on the naive synthetic data - no detected objects.



(e) Pose estimation results for all the detected objects.

(f) Selecting the best 5 poses according to the vector (e_i).

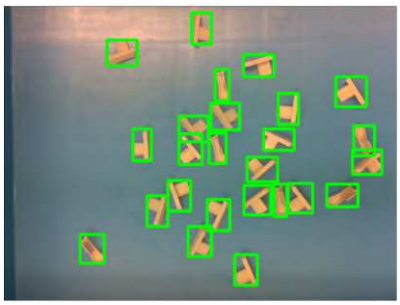
(g) Selecting the best 5 pose estimates according to the scores of Mask R-CNN.



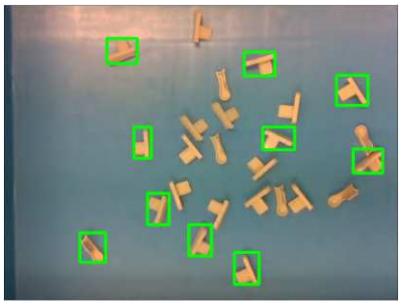
(h) Selecting the best 5 pose estimates according to the cosine similarity values.

Figure 5.9: Qualitative results in a heavily cluttered bin for object 2.

(a) A sample image of a less cluttered bin captured by a Microsoft Azure Kinect RGB-D camera.

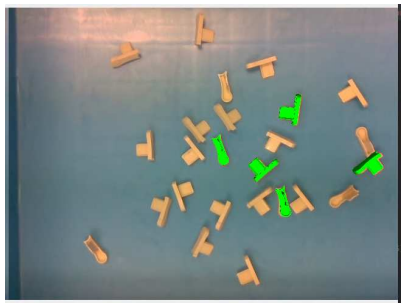


(b) Detection results of Mask R-CNN trained on the realistic synthetic data.

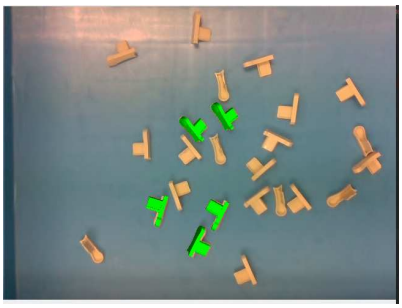


(c) Detection results of Mask R-CNN trained on the naive synthetic data.

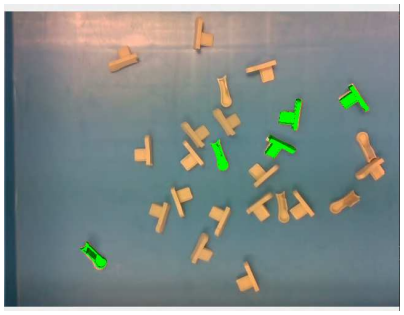
(d) Pose estimation results for all the detected objects.



(e) Selecting the best 5 poses according to the vector (e_i).



(f) Selecting the best 5 pose estimates according to the scores of Mask R-CNN.



(g) Selecting the best 5 pose estimates according to the cosine similarity values.

Figure 5.10: Qualitative results in a less cluttered bin for object 2.

Chapter 6

Prediciting the Probability Distribution on SO(3) Using Implicit Neural Representations

Pose estimation of objects in images is an essential problem in virtual and augmented reality and robotics. Traditional solutions use depth cameras, which can be expensive, and working solutions require long processing times. In this chapter, we present our work HyperPosePDF in Höfer *et al.* (2023), which focuses on the more difficult task when only RGB information is available. To this end, we predict not only the pose of an object but the complete probability density function (pdf) on the rotation manifold. This is the most general way to approach the pose estimation problem and is particularly useful in analysing object symmetries. In this work, we leverage implicit neural representations for the task of pose estimation and show that hypernetworks can be used to predict the rotational pdf. Furthermore, we analyse the Fourier embedding on SO(3) and evaluate the effectiveness of an initial Fourier embedding that proved successful. Our HyperPosePDF outperforms the current state of the art approaches on the SYMSOL (Murphy *et al.* (2021)) dataset.

6.1 Hypernetworks

Hypernetworks are a type of neural network architecture used in deep learning that generate the weights of another neural network, known as the ”target network”. In other words, a hypernetwork takes as input some representation of the desired output network and outputs its weights. Hypernetworks have become very common in deep learning and date back as far as the beginning of the 1990s in the context of meta-learning and self-referential networks (Schmidhuber (1992b)). Several works explored the use of hypernetworks for RNNs, e.g. Schmidhuber (1992a) use fast weights and two feed-forward networks to learn to deal with temporal sequences in this alternative gradient-based system: By producing context-dependent weight changes, the first network will be able to produce very rapid weight changes for the second network. In Ha *et al.* (2016) they look at how hypernetworks can make deep convolutional networks and long recurrent

networks more efficient by incorporating weight-sharing across layers. Their main result is that hypernetworks can generate non-shared weights for LSTMs and achieve near state-of-the-art results on a variety of sequence modelling tasks. Further works that use hypernetworks for RNNs can be found in Gomez and Schmidhuber (2005),Sutskever *et al.* (2011), Ba *et al.* (2016) and Goyal *et al.* (2019). Of course, it is also possible to use hypernetworks for CNNs, which is done by Denil *et al.* (2013), where they measure the percentage of weights that can be predicted without losing accuracy. In their work, they show that they can predict more than 95 % of the weights and only learn a small number of weights without an accuracy drop. Also, in the work by Klein *et al.* (2015), they learn a function that maps the input to filters, which are used for the dynamic convolutional layers. They evaluate their method for the task of short-range weather prediction. Further works on the usage of hypernetworks for CNNs can be found in Jia *et al.* (2016), Bertinetto *et al.* (2016), Perez *et al.* (2018), Kang *et al.* (2017) and Savarese and Maire (2019). Hypernetworks are also used for reinforcement learning (Dwaracherla *et al.* (2020); Huang *et al.* (2021); Sarafian *et al.* (2021)) and architecture search algorithms, which incorporated forms of hypernetworks early on (Stanley *et al.* (2009); Koutnik *et al.* (2010); Brock *et al.* (2017); Zhang *et al.* (2018)). Furthermore, the concept of self-attention can be viewed as a form of adaptive layers (Schlag *et al.* (2021)). Finally, hypernetworks have also been introduced to the field of implicit neural representations (Sitzmann *et al.* (2020a)). However, the use of hypernetworks has mainly been explored for 2D and 3D image and scene generation (Mescheder *et al.* (2019); Littwin and Wolf (2019); Sitzmann *et al.* (2019); Skorokhodov *et al.* (2021); Wang *et al.* (2021b)). We want to apply a hypernetwork to implicit neural representations associated with the task of predicting the probability distribution on the rotation manifold.

6.2 Introduction

Pose estimation has gained an increasing interest in the last years. In many robotic applications, such as object grasping, tracking and occlusion handling, the robotic perception should be able to accurately estimate 3D poses to perform a valid grasp. Traditional approaches assume present depth information and estimate the pose by relying on local invariant features (Abbeloos and Goedemé (2016); Liu *et al.* (2018)) or template-matching (Hinterstoisser *et al.* (2011)). These algorithms rely on expensive evaluations of multiple pose hypotheses, rendering them inefficient. Furthermore, missing textures on many objects hamper their performance.

In our work, we are going to make use of a hypernetwork to predict the weights of an implicit neural representation. This implicit neural representation is associated with the task of representing a probability distribution on SO(3). We then aim to fully utilize the theoretical findings on Fourier embeddings for pose estimation, which have been found to be crucial for the performance of implicit neural representations. We will introduce our approach in the following.

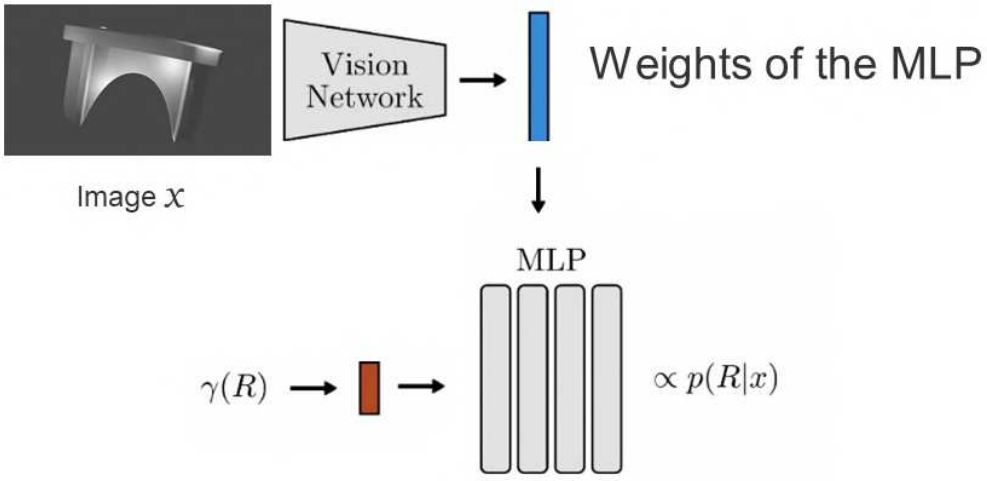


Figure 6.1: Overview of the network architecture. An image x is fed through a vision network that predicts a feature vector. This feature vector is then used as the weights of an MLP. The MLP acts on the rotation manifold and takes as input the Fourier embedded rotation $\gamma(R)$ and outputs its probability of being the underlying rotation in the image.

6.3 Method

Given an image x, our goal is to predict a probability density function

$$p(\cdot|x) : \mathrm{SO}(3) \to [0,1] \tag{6.1}$$

that incorporates not only a single rotation but the general information on the distribution of the rotation of an object in a given image. This is especially helpful in finding symmetry patterns of objects.

We give a general overview of our approach in Figure 6.1. The input image is first fed through a vision network to output a feature vector. This feature vector is then used as the weights of an MLP. The MLP then represents the probability density function on $\mathrm{SO}(3)$ by taking a Fourier-mapped rotation $\mathrm{SO}(3) \ni R \mapsto \gamma(R)$ as input, and outputting the corresponding probability $p(R|x) \in [0,1]$. With this formulation, it is possible to make single pose predictions by taking the mode of the pdf or to predict the full distribution to observe patterns of symmetries.

6.3.1 Fourier Transform on the Rotation Manifold

For an integrable function of the form $f : \mathbb{R} \to \mathbb{C}$ the Fourier transform of f is defined as

$$\mathscr{F}_f(l) = \int_{\mathbb{R}} f(x)\, \mathrm{e}^{-ilx}\, dx. \tag{6.2}$$

The Fourier series is usually applied to periodic, and bounded functions, i.e. of the form $f : [0, 2\pi) \to \mathbb{C}$. Instead of defining f on the range $[0, 2\pi)$, we can also find a mapping between $\alpha \in [0, 2\pi)$ and the rotation matrices $R_\alpha \in \mathrm{SO}(2)$, where α is the rotation angle. This allows us to use the Fourier transform for complex valued functions defined on the rotation group $\mathrm{SO}(2)$. This indeed suggests that the Fourier transform can be generalized to work with various other groups, specifically $\mathrm{SO}(3)$.

In fact, this is possible by introducing the Wigner-D matrices, which are from a technical point of view the irreducible representations of the rotation group $\mathrm{SO}(3)$ (Ping *et al.* (2002)). Leveraging this observation, it is possible to define the Fourier series for a function

$$f : \mathrm{SO}(3) \longrightarrow \mathbb{R}. \tag{6.3}$$

By using the Wigner-D functions $D_l^{m,n}$, which are an orthogonal basis for the rotation group $\mathrm{SO}(3)$, the Fourier series is given as

$$f = \sum_{l=1}^{L} \sum_{m,n=-l}^{l} f_{l,m,n} D_l^{m,n} \tag{6.4}$$

with the integer L denoting the degree of freedom. It is possible to rewrite this into an ordinary Fourier transform by expanding the Wigner-D function to a Fourier sum. In literature, this derivation is usually given by using the Euler angles representation $R(\alpha, \beta, \gamma)$ of the respective rotation. Following section 2.3 it turns out that

$$f(R(\alpha, \beta, \gamma))) = \sum_{l,m,n=-L}^{L} h_l^{m,n} e^{-i\left((m,n,l)(R(\alpha,\beta,\gamma)\right)}, \tag{6.5}$$

where the derivation of the Fourier coefficients $h_l^{m,n}$ can be found in the remainder of section 2.3. For ease of writing, we define $\mathbf{i} := (m, n, l)$. Instead of writing $h_l^{m,n}$ we write $h_{n_1,n_2,n_3} = h_{\mathbf{n}}$. If we define $\mathbf{n} = (n_1, n_2, n_3) = (l, m, n)$ and $\mathbf{x} = (\alpha, \beta, \gamma)$, we can use Lemma 3.3.1 with dimension 3 to find another parameterization of this function, namely

$$f(R) = \sum_{\mathbf{i}=-L, m\geq 0}^{L} a_{\mathbf{i}} \cos(2\pi \mathbf{i} R) + b_{\mathbf{i}} \sin(2\pi \mathbf{i} R), \tag{6.6}$$

where

$$
\begin{aligned}
a_{\mathbf{0}} &= h_{\mathbf{0}}, \\
a_{\mathbf{i}} &= \begin{cases} 0 \;\; \exists j \in \{2,3\} : i_1 = i_{j-1} = 0 \wedge i_j < 0 \\ 2\mathrm{Re}(h_{\mathbf{i}}) \;\; \text{otherwise,} \end{cases} \\
b_{\mathbf{i}} &= \begin{cases} 0 \;\; \exists j \in \{2,3\} : i_1 = i_{j-1} = 0 \wedge i_j < 0 \\ -2\mathrm{Im}(h_{\mathbf{i}}) \;\; \text{otherwise.} \end{cases}
\end{aligned} \tag{6.7}
$$

The main idea is to make the coefficients **a** and **b** trainable by letting them act as weights of a neural network on an initial Fourier embedding. In chapter 3, we showed that for problems of dimension > 2, as in our case, memory problems arise on the current modern Nvidia RTX 2080Ti GPU, if all coefficients are jointly approximated, as the size of the embedding simply gets too large. This introduces the need to find appropriate Fourier embeddings that do not affect the performance and memory consumption of the method. The design choices of the embedding are discussed in the next section.

6.3.2 Fourier Embedding

Recall the embeddings that we introduced in chapter 3. We are going to compare the following three embeddings on a flattened rotation $R \in \mathrm{SO}(3)$, which we call r in the following.

- The positional encoding is defined as:
  $\gamma(r) = [\ldots, \cos(\pi 2^{\frac{j}{m}} r), \sin(\pi 2^{\frac{j}{m}} r), \ldots]$
  for $j = 0, \ldots, m-1$ where $m \in \mathbb{N}$, using a log-linear spacing for each dimension (Mildenhall *et al.* (2020)).

- The Gaussian embedding is defined as:
  $\gamma(r) = [\cos(2\pi \mathbf{B} r), \sin(2\pi \mathbf{B} r)]$, where $\mathbf{B} \in \mathbb{R}^{m \times d}$ is sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$, while σ is the hyperparameter to be optimized Tancik *et al.* (2020).

- The sinusoidal network is defined as: Instead of using an initial Fourier encoding, it is also possible to use a sinusoidal network. Contrary to classical MLPs, it consists of periodic activation functions. It has been shown that an additional initial sinusoidal layer acts as a learnable Fourier embedding layer, achieving a similar or better performance (see chapter 3).

6.4 Experiments

We conduct our experiments on the Symsol I, Symsol II and Pascal3D+ datasets. While we use the common Acc30° metric for Pascal3D+, we evaluate the SYMSOL datasets using two metrics: log likelihood and spread, which we will introduce in the following.

6.4.1 Datasets

SYMSOL I

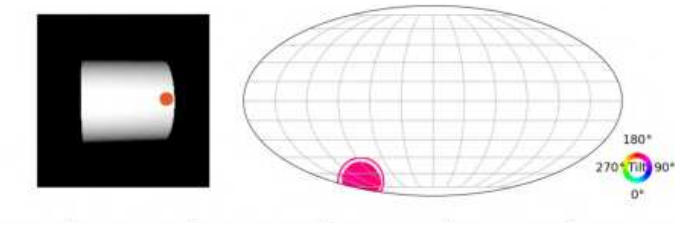
The Symsol I dataset is publicly available as part of the Tensorflow datasets. This dataset is especially interesting as it consists of 5 objects with multiple symmetries, namely: *cone*, *cylinder*, *tetrahedron*, *cube* and *icosahedron*. Here, the *tetrahedron*, *cube* and *icosahedron* have countably many symmetries, i.e. 12, 24 and 60, respectively. As the *cone* and *cylinder* both have continuous symmetries, their annotations are made discrete with an equidistant 1-degree spacing. Each RGB image is of size 224×224. The associated labels per image are its class and the ground truth rotation, including all equivalent rotations.

SYMSOL II

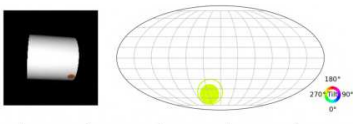
The Symsol II dataset is also publicly available as part of the Tensorflow datasets. This dataset consists of three objects: a *tetrahedron* (*tetX*) with a marked red area, a *cylinder* (*cylO*) with a marked off-center point and a *sphere* (*sphX*) with an X and a marked point. Depending on the visibility, these markings affect the number of symmetries significantly. For example, the sphere without visible markings would have all orientations with the marks on the back possible, but if both markings are visible, the orientation would be unique.

Pascal3D+

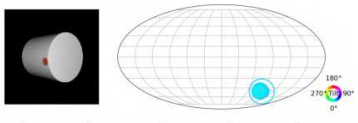
To analyze whether our approach is applicable to single pose estimation, we conduct additional experiments on a subset of the Pascal3D+ dataset Xiang *et al.* (2014). It consists of a subset of the object categories from the well-known PASCAL VOC dataset Everingham *et al.* (2010b), where 3D annotations are added. Furthermore, the dataset has been enlarged by adding more images from the ImageNet dataset Deng *et al.* (2009). The annotation of an object consists of the elevation, azimuth, and distance of the camera position in 3D. With at least 3000 instances per category, it is a challenging dataset of real-world objects, like *planes, trains, bicycles* and more. The choice of a subset is due to the unavailability of an official data loader and existing invalid bounding box annotations in the dataset that we handled individually, e.g. manually adding the missing annotations or skipping elements with incorrect annotations. This leads us to exclude quantitative results of the objects where we can not guarantee alignment with publicly



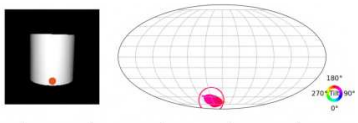
(a) If the red marker is visible, the rotation of the *cylinder* is unique.



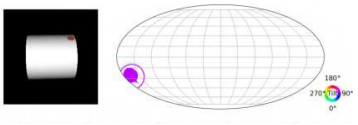
(b) Rotation around the x-axis with the marker being moved to the bottom.



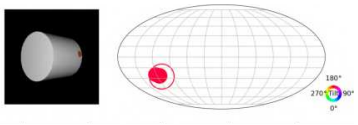
(c) Rotation around the y-axis with the marker being pushed to the front.



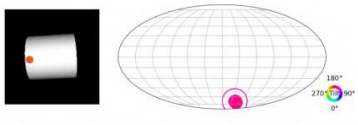
(d) Rotation around the z-axis with the marker being moved to the bottom.



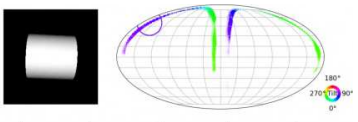
(e) Rotation around the x-axis with the marker being moved to the top.



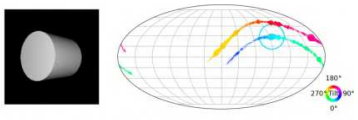
(f) Rotation around the y-axis with the marker being pushed to the back.



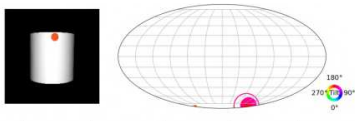
(g) Rotation around the x-axis with the marker being moved to the left.



(h) As the marker is not visible, a continuous symmetry can be seen. Only half of the symmetry axis of a normal *cylinder* is displayed as the model learned to nullify the subspace of rotations for which the marker would be visible.

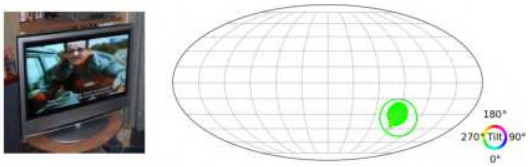


(i) The marker is not visible, therefore our model outputs a continuous symmetry axis with a gap in between representing the area, where the marker would be visible.

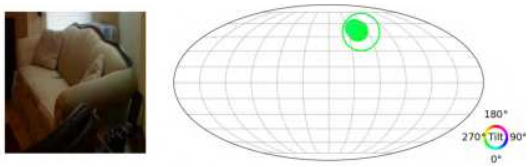


(j) The movement around the z-axis has the effect of maintaining the tilt colour. As in this scenario the marker was always visible, only one unique rotation is present.

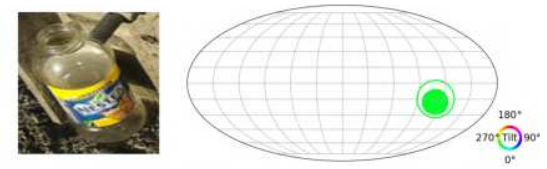
Figure 6.2: Visualization of results on the cylO object from the SYMSOL II dataset. Elements of SO(3) with a positive probability are visualized as points on the grid. Intuitively, we can consider each point on the grid as the direction of a canonical z-axis, and the color indicates the angle of inclination around this axis. Note that the hollow circle indicates the ground truth pose, while the filled area depicts the predicted poses. Note that in the case of a missing red dot, the ground truth pose may be ambiguous and we plot only one possible ground truth pose. The visualization tool was introduced by Murphy *et al.* (2021).



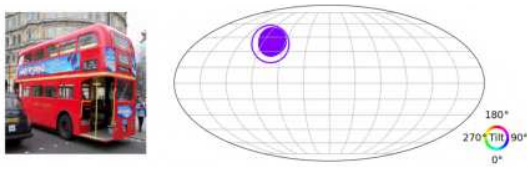
(a) A tv faced towards the camera while a movie is playing.



(b) A sofa in beige with two pillows placed on it.



(c) A bottle with yellow plastic wrapped around it.



(d) A red bus with the front faced to the right.

Figure 6.3: Results on the Pascal3D+ dataset. As all objects in these images are standing upright and are faced towards us, the rotations are closely related. With the presence of texture, symmetries are not existent and hence, we predict only a single rotation for the objects.

available results on this dataset. Still, we show qualitative results in the end of this section. For the train and test splits, we follow the split provided by Liao *et al.* (2019).

6.4.2 Evaluation Metrics

We assume the ground truth labels to be samples from an underlying but unknown distribution, which contains all information about symmetries, noise and ambiguities. As the output of our model is also a distribution, it is standard to compare the two distributions using maximum likelihood. More formally: Our test set consists of images $x \in I$, where each x has annotated poses $\mathbf{R}^{\mathbf{x}} = (R_1^x, \dots, R_k^x)$ for some $k \in \mathbb{N}$ and $k > 1$ if there exist symmetries. We then calculate the averaged log likelihood as follows

$$\mathrm{LL} = \frac{1}{|I|} \sum_{x \in I} \frac{1}{|\mathbf{R}^x|} \sum_{R \in \mathbf{R}^{\mathbf{x}}} \log(p(R|x)).$$

Another way of comparing two distributions is to calculate the spread Spr. It assumes a set of equivalent rotation annotations to be given. It uses the geodesic distance

$$\begin{aligned} d : \mathrm{SO}(3) \times \mathrm{SO}(3) &\to \mathbb{R}_+ \\ (R_1, R_2) &\mapsto \| \log R_1 R_2^T \|_F \end{aligned}$$

using the Frobenius norm $\| \cdot \|_F$. Only the closest ground truth annotation is then taken

	SYMSOL I (log likelihood ↑)					
	cone	cyl.	tet.	cube	ico.	avg.
Deng *et al.* (2022)	0.16	-0.95	0.27	-4.44	-2.45	-1.48
Gilitschenski *et al.* (2019)	3.84	0.88	-2.29	-2.29	-2.29	-0.43
Prokudin *et al.* (2018)	-1.87	-3.34	-1.28	-1.86	-0.50	-2.39
Murphy *et al.* (2021)	4.45	4.26	5.70	4.81	1.28	4.10
HyperPosePDF (ours)	**5.74**	**4.73**	**7.04**	**6.77**	**5.10**	**5.78**

Table 6.1: A model was jointly trained for all of the SYMSOL I classes. We compare our results against multimodal mixture models (Deng *et al.* (2022); Gilitschenski *et al.* (2019); Prokudin *et al.* (2018)) and Implicit-PDF by Murphy *et al.* (2021) which we all outperform by a significant amount in the log likelihood metric. A value of -2.29 represents the minimal information of a uniform distribution on SO(3).

	SYMSOL I (Spread ↓)					
	cone	cyl.	tet.	cube	ico.	avg.
Deng *et al.* (2022)	10.1	15.2	16.7	40.7	28.5	22.2
Murphy *et al.* (2021)	1.4	1.4	4.6	4.0	8.4	4.0
HyperPosePDF (ours)	**0.6**	**0.5**	**3.3**	**2.2**	**3.2**	**1.97**

Table 6.2: Similar to Table 6.1 we train a joint model for all objects in the SYMSOL I dataset and compare it to the method of Deng *et al.* (2022) and Implicit-PDF by Murphy *et al.* (2021). For the cone and cylinder, the spread of the probability prediction away from the rotational continuous symmetry has a value of less than one degree.

into account

$$Spr = \mathrm{E}_{R\sim p(R|x)}[\min_{R'\in\mathbf{R}^x} d(R,R')].$$

6.4.3 Experiments on the SYMSOL I Dataset

Our implementation specifics are as follows. For our vision module, we use a pretrained ResNet-50 backbone. We predict the weights of a one-layer network with a width of 256. The number of coefficients used for the positional encoding is set to 4. A learning rate of $1e-4$ is used for the first 1000 iterations; then, a cosine decay is applied. Using the Adam optimizer, we evaluate our model after 200k iterations using a batch size of 16.

The model jointly learns all object classes of the SYMSOL I dataset. Table 6.1 shows the log-likelihood results. In this metric, we can demonstrate superior results to compet-

	SYMSOL I (log likelihood ↑)					
	cone	cyl.	tet.	cube	ico.	avg.
Positional Encoding	5.74	4.73	7.04	6.77	5.10	5.78
Gaussian encoding	**5.78**	**5.05**	7.16	6.80	**5.48**	**6.05**
Siren encoding	5.66	4.71	**8.06**	**7.34**	4.01	5.96

Table 6.3: We evaluate the effect of an initial Fourier embedding being applied to our network. In this table we compare the effect of positional encoding (Mildenhall *et al.* (2020)) vs. Gaussian encoding (Tancik *et al.* (2020)) vs. a learnable sinusoidal layer (Sitzmann *et al.* (2020a)). While the positional encoding is the most spread embedding, it is possible to increase the performance by changing to a Gaussian embedding or a learnable sinusoidal layer. For the experiments reported in the other tables, we use a positional encoding.

ing methods on all objects individually and on average. This is particularly visible for the objects *cone, tetrahedron, cube* and *icosahedron*. We were able to rerun the experiments of the competing methods and receive numbers close to their official numbers. Still, we show their reported values in our table. Note that we used the positional encoding in this experiment. We can further improve the performance by switching to Gaussian or Siren encodings. Table 6.2 shows the spread results. We compare against reported values from Deng *et al.* (2022) and Murphy *et al.* (2021). The metric values are in degrees and show how well the method is able to capture the ground truths. For the cone and cylinder, the spread of the probability prediction away from the continuous rotational symmetry has a value of less than one degree. The spread experiments have only been conducted on the SYMSOL I dataset as it is the only one with full symmetry annotations. If only a single ground truth is known, this metric would be misleading as it penalizes correct predictions if no corresponding annotation is available.

We compare the different Fourier embeddings as introduced in section 3.2. We found a scale of 2 to perform best for the Gaussian embedding. Likewise, the performance of the sinusoidal embedding heavily depends on the chosen bias, which we found to perform best with a value of 1. Table 6.3 shows that, in general, an embedding is helpful, and with proper parameters, it is possible for the Gaussian and Siren embedding to outperform the positional encoding.

6.4.4 Experiments on the SYMSOL II Dataset

We took the same implementation specifics as for the SYMSOL I dataset. Following Murphy *et al.* (2021), we trained a network for each object separately. Table 6.4 shows that we are able to achieve promising results on this challenging dataset. In particular, we show in the experiments that our method is able to represent distributions that cannot

	SYMSOL II (log likelihood ↑)			
	sphX	cylO	tetX	avg.
Deng *et al.* (2022)	1.12	2.99	3.61	2.57
Gilitschenski *et al.* (2019)	3.32	4.88	2.90	3.70
Prokudin *et al.* (2018)	4.19	4.16	1.48	0.48
Murphy *et al.* (2021)	7.30	6.91	8.49	7.57
HyperPosePDF (Ours)	**7.73**	**7.12**	**8.53**	**7.72**

Table 6.4: For this experiment, we trained a model for each object of the SYMSOL II dataset separately and compare our results against multimodal mixture models (Deng *et al.* (2022); Gilitschenski *et al.* (2019); Prokudin *et al.* (2018)) and Implicit-PDF by Murphy *et al.* (2021). We are able to achieve better results than our competitors on all objects. These experiments were especially challenging due to the differing numbers of symmetries that are dependant on the visibility of the markers on the objects.

be well approximated by mixture-based models. That is mainly because of the changing amount of present symmetries due to the visibility of the given markings.

In general, it is not clear how to visualize a pose. Reporting the values of a 3×3 rotation matrix will not help the reader to check whether the predicted pose is correct or not. Just recently in Murphy *et al.* (2021) a new method for visualizing poses was introduced. With the help of Hopf fibrations, they project circles of poses from SO(3) to the 2-sphere and then use the color to indicate the location on the circle. Because of the projection to a lower dimension, limitations do exist. Still, we are happy to use the visualization tool to demonstrate the performance of our model. Figure 6.5 shows qualitative results of a model trained on SYMSOL II. We plot the ground truth and predicted poses of *cylO* object. The plots illustrate that the model has successfully learned the pose distribution of this object. When the red dot is visible, the model successfully collapses the distribution to predict a small range of poses. When it is not visible, the model outputs a smooth distribution of all possible poses given how the object is visible in the figure.

6.4.5 Experiments on the Pascal3D+ Dataset

Our implementation specifics are as follows. As the complexity of the images in the Pascal3D+ dataset is higher than in the SYMSOL dataset, we choose a larger pretrained ResNet-101 backbone for our vision module. We predict the weights of a one-layer network with a width of 256. Using the Adam optimizer, we evaluate our model after 150k iterations using a batch size of 16. A learning rate of $1e-5$ is used for the first 1000 iterations, and then a cosine decay is applied.

Table 6.5 shows our evaluations in the standard Acc30° metric and the median angular error. While our method is specifically designed to account for present symmetries, the

	PASCAL3D+ (Acc30° ↑)					
	bottle	bus	table	sofa	tv	avg
Liao *et al.* (2019)	0.93	0.95	0.61	0.95	0.82	0.852
Mohlin *et al.* (2020)	0.94	0.95	0.62	0.85	0.84	0.840
Prokudin *et al.* (2018)	0.96	0.93	0.76	0.90	0.91	0.892
Tulsiani and Malik (2015)	0.93	0.98	0.62	0.82	0.80	0.830
Mahendran *et al.* (2018)	0.96	0.97	0.67	0.97	0.88	0.890
Murphy *et al.* (2021)	0.93	0.95	0.78	0.88	0.86	0.880
HyperPosePDF (Ours)	0.83	0.92	0.97	0.89	0.88	0.898

	PASCAL3D+ (Median ↓)					
	bottle	bus	table	sofa	tv	avg
Liao *et al.* (2019)	10.3	4.8	12.0	12.3	14.3	10.74
Mohlin *et al.* (2020)	7.8	3.3	12.5	13.8	11.7	9.82
Prokudin *et al.* (2018)	5.4	2.9	12.6	9.1	12.0	8.4
Tulsiani and Malik (2015)	12.9	5.8	15.2	13.7	15.4	12.6
Mahendran *et al.* (2018)	7.0	3.1	11.3	10.2	11.7	8.66
Murphy *et al.* (2021)	8.8	3.4	7.3	9.5	12.3	8.26
HyperPosePDF (Ours)	11.7	3.9	4.2	5.8	6.5	6.42

Table 6.5: Results on objects from the Pascal3D+ dataset. A single model was jointly trained on all classes. We compare our results in the Acc30° and the median in degrees. We are able to achieve similar or slightly better results than the competing methods.

table shows that we are also competitive in the task of single-pose prediction. In Liao *et al.* (2019) the authors reported values that are incorrectly lowered by a factor of $\sqrt{2}$. Hence we report the corrected values in our experiments. Visualizations can be found in Figure 6.4, where we display four objects: a bottle, a sofa, a bus, and a tv monitor. With the presence of textures, the pose predictions are unique.

6.5 Conclusion

Previous works demonstrated that hypernetworks can be used to predict implicit neural representations for the task of 2D and 3D shape reconstruction. To the best of our knowledge, we are the first to show that hypernetworks are able to predict the weights of an implicit neural representation associated with the task of pose estimation. HyperPosePDF is able to predict a non-parametric distribution on the rotation manifold, designed to incorporate the uncertainty of symmetry, noise and ambiguities. Additionally, we could show that the commonly used Fourier embedding for INRs is also capable

of boosting the pose estimation results. Furthermore, we achieve superior performance on the challenging SYMSOL datasets that consist of objects with varying symmetries. Besides that, we are able to maintain comparable performance on single-pose estimation evaluated on the Pascal3D+ dataset.

This work demonstrated promising results in pose estimation tasks. In future works, it would be interesting to see how these insights generalize to new application domains, such as spin detection in table tennis robots or visual-inertial odometry in flying robots.

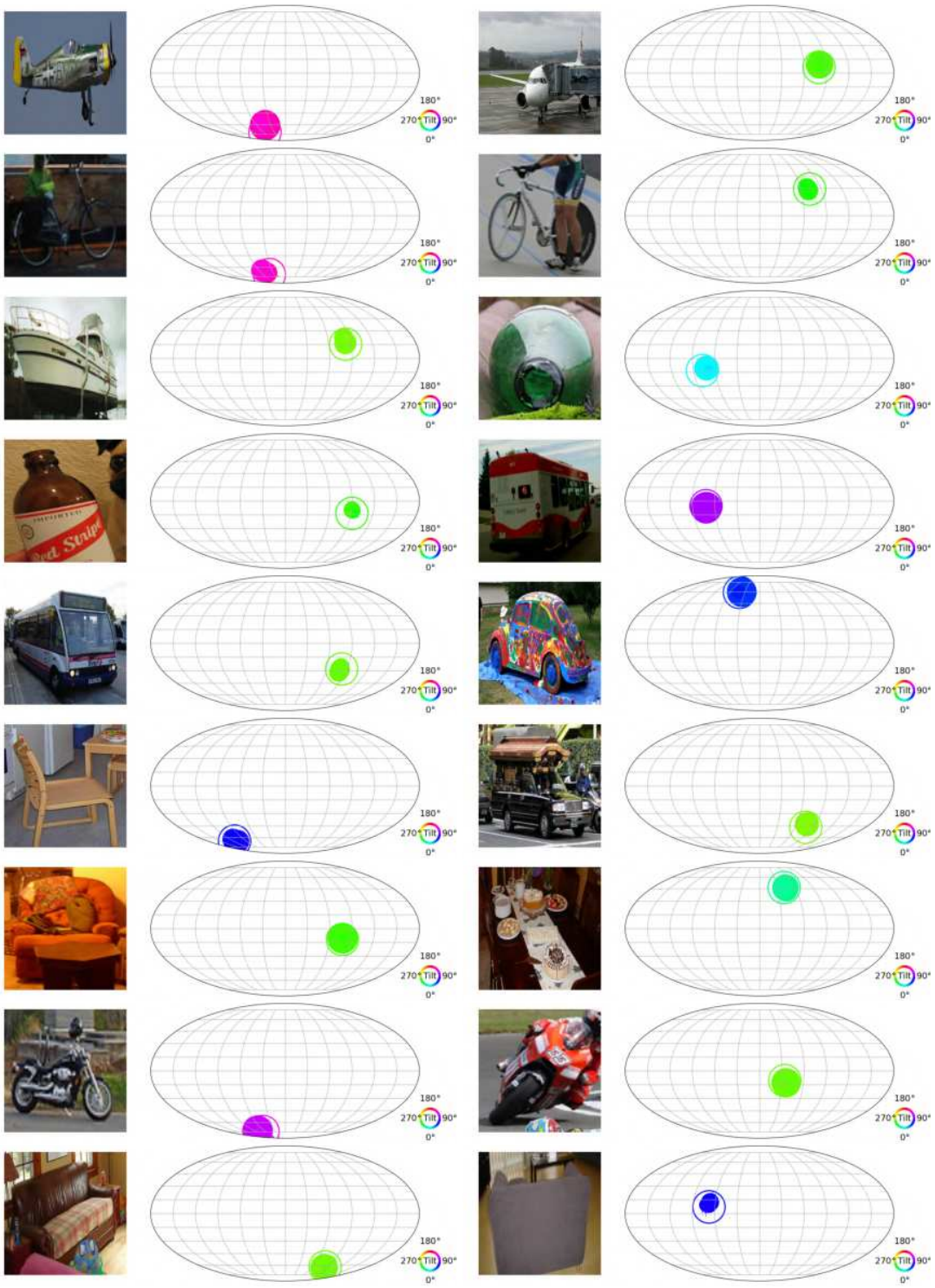
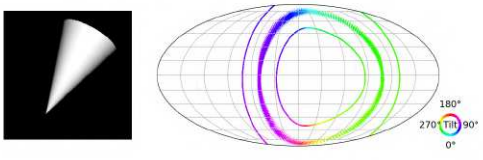
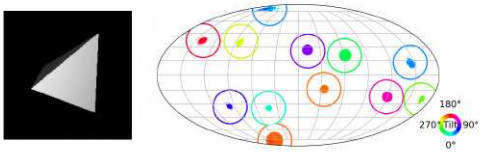


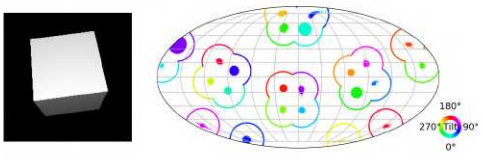
Figure 6.4: Further results on daily life images from the Pascal Voc dataset.



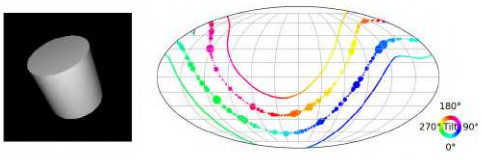
(a) No marking on the object, therefore our model predicts a continous symmetry.



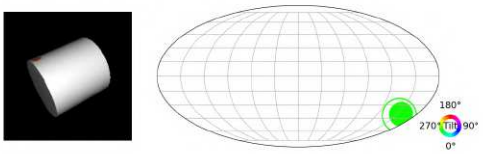
(b) Our model captures all 12 symmetries.



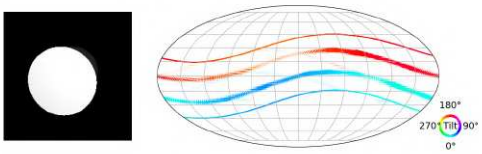
(c) The cube consists of 24 symmetries which we are able to capture.



(d) The cylinder without marker has a continous symmetry.



(e) The red marker is visible on the cylinder. Hence, no symmetries are present.



(f) The cylinder without a marker has a continuous symmetry.

Figure 6.5: Visualization of results on objects from the SYMSOL and SYMSOL II datasets.

Chapter 7

Overall Conclusion

In this dissertation, several research areas in the field of computer vision were presented. These ranged from object detection and segmentation to implicit neural representations, meta learning and pose estimation. Overall, we presented two implicit methods for pose estimation based on RGB data and one of them we embedded in a complete framework from data generation to a pose filtering algorithm.

Beginning in Chapter 2, we concentrated on the introduction of the basics of Fourier Theory and rotation representations. In this context, we started with the introduction of rotation parameterizations as well as the subsequent discussion about their usability for pose estimation. At the same time, Fourier Theory is a core element of implicit neural representations, and is therefore also introduced here. As a conclusion of this chapter, Fourier Theory was discussed on SO(3).

In Chapter 3 we first introduced the research field of implicit neural representations. In this area neural networks are used to approximate complex functions. These are able to parametrize discrete signal representations by a continuous function. In this chapter, we addressed the tasks of image regression and the novel view synthesis task. We put a special focus on an initial Fourier embedding, which is crucial for this task. Specifically, we introduced an integer embedding that is equivalent to the Fourier series for a perceptron, provided that the width of this layer is large enough. At the same time we could show that a layer with a sine activation function is equivalent to a learnable Fourier embedding. With these results, we were able to beat other well-known embeddings such as positional and gaussian embedding. In addition, we introduced in this chapter a method to gradually improve an initially poorly chosen embedding by pruning unimportant elements of the embedding and replacing them with new elements that have a positive impact on the standard deviation of the embedding matrix.

The foundations for object localization are laid in chapter 4. Starting with an evaluation of the framerate of state of the art object detectors on a GPU/CPU and mobile GPU, we continue with the introduction of object segmentation. In this sense, we presented FourierMask, a method for instance segmentation that builds on Mask R-CNN and uses implicit neural representations in a special way to improve the mask at the boundary. The idea was the following, first a coarse mask is determined that in principle matches the baseline Mask R-CNN. It is typically able to make out the object, but usually does

not hit the exact contours of details, like the exact segmentation of the hands or other small details. Now, however, a more accurate prediction is made at the edge pixels of the segmentation mask, where the confidence score is close to 0.5. For this purpose, an implicit neural representation takes on the task of processing these pixels using a Fourier representation to ultimately determine whether these pixels belong to the object or not.

Chapter 5 now introduced the task of pose estimation. After a short overview of techniques and problems we introduced a framework for pose estimation in highly cluttered bin picking scenarios. Here, we started by generating synthetic data on which we could train our detectors. Our pipeline consisted of Mask R-CNN for object localization, followed by an implicit pose estimator, which we called augmented autoencoder. This was trained to isolate objects from their background on RGB data and generate latent code. This latent code was then compared to previously generated latent codes that implicitly represented a discrete subset of SO(3). The rotation with the most similar latent code was then taken as the estimate. Since in our case there were several hundred objects in a cluster, we introduced a method for filtering the best estimates based on depth data.

Finally, in Chapter 6, we introduced HyperPosePDF. Our network HyperPosePDF worked as follows: we first used a vision network, which takes RGB images as input and then determines weights for a second network. In other words, it takes on the function of a hypernetwork. The second network is an implicit neural representation, which gets Fourier mapped rotations as input and maps them to a probability. This gives a probability distribution on SO(3) for the pose of the given object, which allows to generate multiple and even continuous pose hypotheses, particularly well suited for symmetric objects. Compared to the mixture based models, our method is more effective in capturing uncertainties.

Abbreviations

AP	Average Precision
AR	Average Recall
CAD	Computer-Aided Design
CNN	Convolutional Neural Network
COCO	Common Objects in Context
FCOS	Fully Convolutional Object Detector
FF	Fourier Features
FFT	Fast Fourier Transform
FM	FourierMask
FNN	Fourier Neural Networks
GPU	Graphics Processing Unit
ICP	Iterative Closest Point
IoU	Intersection over Union
INR	Implicit Neural Representation
k-NN	k-Nearest Neighbors
LSTM	Long Short-Term Memory
mAP	Mean Average Precision
MSPD	Maximum Symmetry-aware Projection Distance
MSSD	Maximum Symmetry-aware Surface Distance
NeRF	Neural Radiance Fields
NN	Neural Network
PDF	Probability Density Function
PSNR	Peak Signal-to-Noise Ratio
PT	Progressive Training
R-CNN	Region Based Convolutional Neural networks
ReLu	Rectified Linear Unit
SIREN	SInusoidal REpresentation Networks
SOTA	State of the Art
SSD	Single Shot Detector
SYMSOL	Symmetric Solids
VSD	Visual Surface Discrepancy
YOLACT	You Only Look At CoefficienTs
YOLO	You Only Look Once

Bibliography

Abbeloos, W. and Goedemé, T. (2016). Point pair feature based object detection for random bin picking. In *2016 13th Conference on Computer and Robot Vision (CRV)*, pages 432–439. IEEE.

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.

Ba, J., Hinton, G. E., Mnih, V., Leibo, J. Z., and Ionescu, C. (2016). Using fast weights to attend to the recent past. *Advances in neural information processing systems*, **29**.

Balntas, V., Doumanoglou, A., Sahin, C., Sock, J., Kouskouridas, R., and Kim, T.-K. (2017). Pose guided rgbd feature learning for 3d object pose estimation. In *Proceedings of the IEEE international conference on computer vision*, pages 3856–3864.

Bay, H., Tuytelaars, T., and Gool, L. V. (2006). Surf: Speeded up robust features. In *European conference on computer vision*, pages 404–417. Springer.

Benbarka & Höfer, N. . 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.

Bertinetto, L., Valmadre, J., Henriques, J. F., Vedaldi, A., and Torr, P. H. (2016). Fully-convolutional siamese networks for object tracking. In *European conference on computer vision*, pages 850–865. Springer.

Brock, A., Lim, T., Ritchie, J. M., and Weston, N. (2017). Smash: one-shot model architecture search through hypernetworks. *arXiv preprint arXiv:1708.05344*.

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.

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, B., Girshick, R., Dollár, P., Berg, A. C., and Kirillov, A. (2021). Boundary IoU: Improving object-centric image segmentation evaluation. In *CVPR*.

Chirikjian, G. S. (2000). *Engineering applications of noncommutative harmonic analysis: with emphasis on rotation and motion groups*. CRC press.

Codebasics (2020). Image classification vs object detection vs image segmentation — deep learning tutorial 28. `https://www.youtube.com/watch?v=taC5pMCm70U`.

Corona, E., Kundu, K., and Fidler, S. (2018). Pose estimation for objects with rotational symmetry. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 7215–7222. IEEE.

Deitmar, A. and Echterhoff, S. (2014). *Principles of harmonic analysis*. Springer.

Deng, B., Lewis, J. P., Jeruzalski, T., Pons-Moll, G., Hinton, G., Norouzi, M., and Tagliasacchi, A. (2020a). Nasa neural articulated shape approximation. In *European Conference on Computer Vision*, pages 612–628. Springer.

Deng, H., Bui, M., Navab, N., Guibas, L., Ilic, S., and Birdal, T. (2022). Deep bingham networks: Dealing with uncertainty and ambiguity in pose estimation. *International Journal of Computer Vision*, pages 1–28.

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.

Deng, X., Xiang, Y., Mousavian, A., *et al.* (2020b). Self-supervised 6d object pose estimation for robot manipulation. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3665–3671. IEEE.

Denil, M., Shakibi, B., Dinh, L., Ranzato, M., and De Freitas, N. (2013). Predicting parameters in deep learning. *Advances in neural information processing systems*, **26**.

Denninger, M., Sundermeyer, M., Winkelbauer, D., Zidan, Y., and Olefir, D. (2019). Blenderproc. *arXiv preprint arXiv:1911.01911*.

Do, T.-T., Cai, M., Pham, T., and Reid, I. (2018). Deep-6dpose: Recovering 6d object pose from a single rgb image. *arXiv preprint arXiv:1802.10367*.

Drost, B., Ulrich, M., Navab, N., and Ilic, S. (2010). Model globally, match locally: Efficient and robust 3d object recognition. In *2010 IEEE computer society conference on computer vision and pattern recognition*, pages 998–1005. Ieee.

Dwaracherla, V., Lu, X., Ibrahimi, M., Osband, I., Wen, Z., and Van Roy, B. (2020). Hypermodels for exploration. *arXiv preprint arXiv:2006.07464*.

Everingham, M., Van Gool, L., Williams, C. K. I., Winn, J., and Zisserman, A. (2010a). The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, **88**(2), 303–338.

Everingham, M., Van Gool, L., Williams, C. K., Winn, J., and Zisserman, A. (2010b). The pascal visual object classes (voc) challenge. *International journal of computer vision*, **88**(2), 303–338.

Everingham, M., Eslami, S. M. A., and others. (2015). The pascal visual object classes challenge: A retrospective. *International Journal of Computer Vision*, **111**(1), 98–136.

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

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.

Gilitschenski, I., Sahoo, R., Schwarting, W., Amini, A., Karaman, S., and Rus, D. (2019). Deep orientation uncertainty learning based on a bingham loss. In *International Conference on Learning Representations*.

Girshick, R. (2015). Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448.

Girshick, R., Donahue, J., Darrell, T., and Malik, J. (2014). Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587.

Goesele, M., Snavely, N., Curless, B., Hoppe, H., and Seitz, S. M. (2007). Multi-view stereo for community photo collections. In *2007 IEEE 11th International Conference on Computer Vision*, pages 1–8. IEEE.

Gomez, F. and Schmidhuber, J. (2005). Evolving modular fast-weight networks for control. In *International Conference on Artificial Neural Networks*, pages 383–389. Springer.

Goyal, A., Lamb, A., Hoffmann, J., Sodhani, S., Levine, S., Bengio, Y., and Schölkopf, B. (2019). Recurrent independent mechanisms. *arXiv preprint arXiv:1909.10893*.

Ha, D., Dai, A., and Le, Q. V. (2016). Hypernetworks. *arXiv preprint arXiv:1609.09106*.

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 (CVPR)*.

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.

Hemingway, E. G. and O'Reilly, O. M. (2018). Perspectives on euler angle singularities, gimbal lock, and the orthogonality of applied forces and applied moments. *Multibody System Dynamics*, **44**(1), 31–56.

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.

Hinterstoisser, S., Holzer, S., *et al.* (2011). Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *2011 international conference on computer vision*, pages 858–865. IEEE.

Hinterstoisser, S., Lepetit, V., Ilic, S., Holzer, S., Bradski, G., Konolige, K., and Navab, N. (2012). Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer.

Hodan, T., Haluza, P., *et al.* (2017). T-less: An rgb-d dataset for 6d pose estimation of texture-less objects. In *2017 IEEE Winter Conference on Applications of Computer Vision (WACV)*, pages 880–888. IEEE.

Hodaň, T., Sundermeyer, M., Drost, B., Labbé, Y., Brachmann, E., Michel, F., Rother, C., and Matas, J. (2020). Bop challenge 2020 on 6d object localization. In *European Conference on Computer Vision*, pages 577–594. Springer.

Höfer, T. and Zell, A. (2022). Automatic adjustment of fourier embedding parameterizations. In *2022 27th International Conference on Pattern Recognition (ICPR)*, pages 7833–7840. IEEE.

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.

Höfer, T., Kiefer, B., and Zell, A. (2023). Hyperposepdf: Predicting the probability distribution on so(3). In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pages 2041–2050.

Huang, Y., Xie, K., Bharadhwaj, H., and Shkurti, F. (2021). Continual model-based reinforcement learning with hypernetworks. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 799–805. IEEE.

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.

Jia, X., De Brabandere, B., Tuytelaars, T., and Gool, L. V. (2016). Dynamic filter networks. *Advances in neural information processing systems*, **29**.

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.

Joffe, B., Walker, T., Gourdon, R., and Ahlin, K. (2019). Pose estimation and bin picking for deformable products. *IFAC-PapersOnLine*, **52**(30), 361–366.

Kang, D., Dhar, D., and Chan, A. (2017). Incorporating side information by adaptive convolution. *Advances in Neural Information Processing Systems*, **30**.

Kehl, W., Milletari, F., Tombari, F., Ilic, S., and Navab, N. (2016). Deep learning of local rgb-d patches for 3d object detection and 6d pose estimation. In *European conference on computer vision*, pages 205–220. Springer.

Kehl, W., Manhardt, F., Tombari, F., Ilic, S., and Navab, N. (2017). Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1521–1529.

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*, pages 9799–9808.

Klein, B., Wolf, L., and Afek, Y. (2015). A dynamic convolutional layer for short range weather prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4840–4848.

Konishi, Y., Hanzawa, Y., Kawade, M., and Hashimoto, M. (2016). Fast 6d pose estimation from a monocular image using hierarchical pose trees. In *European Conference on Computer Vision*, pages 398–413. Springer.

Koutnik, J., Gomez, F., and Schmidhuber, J. (2010). Evolving neural networks in compressed weight space. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, pages 619–626.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). 2012 alexnet. *Adv. Neural Inf. Process. Syst*, pages 1–9.

Kuo, W., Angelova, A., Malik, J., and Lin, T.-Y. (2019). Shapemask: Learning to segment novel objects by refining shape priors. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9207–9216.

Labbé, Y., Carpentier, J., Aubry, M., and Sivic, J. (2020). Cosypose: Consistent multi-view multi-object 6d pose estimation. In *European Conference on Computer Vision*, pages 574–591. Springer.

Lee, Y. and Park, J. (2020). Centermask: Real-time anchor-free instance segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 13906–13915.

Li, Z., Wang, G., and Ji, X. (2019). Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7678–7687.

Liang, J., Homayounfar, N., Ma, W.-C., Xiong, Y., Hu, R., and Urtasun, R. (2020). Polytransform: Deep polygon transformer for instance segmentation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9131–9140.

Liao, S., Gavves, E., and Snoek, C. G. (2019). Spherical regression: Learning viewpoints, surface normals and 3d rotations on n-spheres. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 9759–9767.

Lin, C.-H., Ma, W.-C., Torralba, A., and Lucey, S. (2021). Barf: Bundle-adjusting neural radiance fields. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 5741–5751.

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

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., Dollar, P., Girshick, R., He, K., Hariharan, B., and Belongie, S. (2017b). Feature pyramid networks for object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Littwin, G. and Wolf, L. (2019). Deep meta functionals for shape representation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1824–1833.

Liu, D., Arai, S., Miao, J., *et al.* (2018). Point pair feature-based pose estimation with multiple edge appearance models (ppf-meam) for robotic bin picking. *Sensors*, **18**(8), 2719.

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., 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, W., Anguelov, D., Erhan, D., *et al.* (2016). Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, **60**(2), 91–110.

Mahendran, S., Ali, H., and Vidal, R. (2018). A mixed classification-regression framework for 3d pose estimation from 2d images. *arXiv preprint arXiv:1805.03225*.

Malan, D. (2004). Representation of how the axis and angle rotation representation can be visualized. Public Domain.

Manhardt, F., Arroyo, D. M., Rupprecht, C., Busam, B., Birdal, T., Navab, N., and Tombari, F. (2019). Explaining the ambiguity of object detection and 6d pose from visual data. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 6841–6850.

Marcel, S. and Rodriguez, Y. (2010). Torchvision the machine-vision package of torch. In *Proceedings of the 18th ACM international conference on Multimedia*, pages 1485–1488.

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.

Mohlin, D., Sullivan, J., and Bianchi, G. (2020). Probabilistic orientation estimation with matrix fisher distributions. *Advances in Neural Information Processing Systems*, **33**, 4884–4893.

Mur-Artal, R., Montiel, J. M. M., and Tardos, J. D. (2015). Orb-slam: a versatile and accurate monocular slam system. *IEEE transactions on robotics*, **31**(5), 1147–1163.

Murphy, K., Esteves, C., Jampani, V., Ramalingam, S., and Makadia, A. (2021). Implicit-pdf: Non-parametric representation of probability distributions on the rotation manifold. *arXiv preprint arXiv:2106.05965*.

Murray, R. M., Li, Z., and Sastry, S. S. (2017). *A mathematical introduction to robotic manipulation*. CRC press.

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. (2019). Texture fields: Learning texture representations in function space. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4531–4540.

Oppenheim, A. V., Buck, J., Daniel, M., Willsky, A. S., Nawab, S. H., and Singer, A. (1997). *Signals & systems*. Pearson Educación.

Park, J. J., Florence, P., Straub, J., Newcombe, R., and Lovegrove, S. (2019a). 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.

Park, K., Patten, T., and Vincze, M. (2019b). Pix2pose: Pixel-wise coordinate regression of objects for 6d pose estimation. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7668–7677.

Pashevich, A., Strudel, R., Kalevatykh, I., Laptev, I., and Schmid, C. (2019). Learning to augment synthetic images for sim2real policy transfer. In *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2651–2657. IEEE.

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.

Peretroukhin, V., Giamou, M., Rosen, D., and Greene, W. (2020). A smooth representation of belief of so (3) for deep rotation learning with uncertainty. RSS.

Perez, E., Strub, F., De Vries, H., Dumoulin, V., and Courville, A. (2018). Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32.

Ping, J., Wang, F., and Chen, J.-Q. (2002). *Group representation theory for physicists*. World Scientific Publishing Company.

Pitteri, G., Ramamonjisoa, M., Ilic, S., and Lepetit, V. (2019). On object symmetries and 6d pose estimation from images. In *2019 International Conference on 3D Vision (3DV)*, pages 614–622. IEEE.

Potts, D., Prestin, J., and Vollrath, A. (2007). A fast fourier algorithm on the rotation group. *Preprint A-07-06, Univ. zu Lübeck*.

Prestin, J. (2010). The nonequispaced fast so (3) fourier transform, generalisations and applications.

Prokudin, S., Gehler, P., and Nowozin, S. (2018). Deep directional statistics: Pose estimation with uncertainty quantification. In *Proceedings of the European conference on computer vision (ECCV)*, pages 534–551.

Rad, M. and Lepetit, V. (2017). Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Proceedings of the IEEE international conference on computer vision*, pages 3828–3836.

Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788.

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, Hamd ul Moqeet, B., Nuri, Höfer, T., and Zell, A. (2022). Fouriermask: Instance segmentation using fourier mapping in implicit neural networks. In *International Conference on Image Analysis and Processing*, pages 587–598. Springer.

Riaz, H. U. M., Benbarka, N., and Zell, A. (2021). Fouriernet: Compact mask representation for instance segmentation using differentiable shape decoders. In *2020 25th International Conference on Pattern Recognition (ICPR)*, pages 7833–7840. IEEE.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., *et al.* (2015). Imagenet large scale visual recognition challenge. *International journal of computer vision*, **115**(3), 211–252.

Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE international conference on robotics and automation*, pages 3212–3217. IEEE.

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.

Sarafian, E., Keynan, S., and Kraus, S. (2021). Recomposing the reinforcement learning building blocks with hypernetworks. In *International Conference on Machine Learning*, pages 9301–9312. PMLR.

Savarese, P. and Maire, M. (2019). Learning implicitly recurrent cnns through parameter sharing. *arXiv preprint arXiv:1902.09701*.

Schlag, I., Irie, K., and Schmidhuber, J. (2021). Linear transformers are secretly fast weight programmers. In *International Conference on Machine Learning*, pages 9355–9366. PMLR.

Schmidhuber, J. (1992a). Learning to control fast-weight memories: An alternative to dynamic recurrent networks. *Neural Computation*, **4**(1), 131–139.

Schmidhuber, J. (1992b). Steps towardsself-referential'neural learning: A thought experiment.

Schönberger, J. L., Zheng, E., Frahm, J.-M., and Pollefeys, M. (2016). Pixelwise view selection for unstructured multi-view stereo. In *European conference on computer vision*, pages 501–518. Springer.

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. (2020a). Implicit neural representations with periodic activation functions. *Advances in Neural Information Processing Systems*, **33**.

Sitzmann, V., Martel, J. N., Bergman, A. W., Lindell, D. B., and Wetzstein, G. (2020b). Implicit neural representations with periodic activation functions. In *Proc. NeurIPS*.

Skorokhodov, I., Ignatyev, S., and Elhoseiny, M. (2021). Adversarial generation of continuous images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10753–10764.

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

Stanley, K. O., D'Ambrosio, D. B., and Gauci, J. (2009). A hypercube-based encoding for evolving large-scale neural networks. *Artificial life*, **15**(2), 185–212.

Sundermeyer, M., Marton, Z.-C., Durner, M., and Triebel, R. (2020). Augmented autoencoders: Implicit 3d orientation learning for 6d object detection. *International Journal of Computer Vision*, **128**(3), 714–729.

Sutskever, I., Martens, J., and Hinton, G. E. (2011). Generating text with recurrent neural networks. In *ICML*.

Tancik, M., Srinivasan, P., Mildenhall, B., Fridovich-Keil, S., Raghavan, N., Singhal, U., Ramamoorthi, R., Barron, J., and Ng, R. (2020). Fourier features let networks learn high frequency functions in low dimensional domains. *Advances in Neural Information Processing Systems*, **33**, 7537–7547.

Tejani, A., Tang, D., Kouskouridas, R., and Kim, T.-K. (2014). Latent-class hough forests for 3d object detection and pose estimation. In *European Conference on Computer Vision*, pages 462–477. Springer.

Tian, Z., Shen, C., Chen, H., and He, T. (2019). Fcos: Fully convolutional one-stage object detection. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 9627–9636.

Tobin, J., Fong, R., Ray, A., *et al.* (2017). Domain randomization for transferring deep neural networks from simulation to the real world. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 23–30. IEEE.

Tombari, F., Salti, S., and Stefano, L. D. (2010). Unique signatures of histograms for local surface description. In *European conference on computer vision*, pages 356–369. Springer.

Tulsiani, S. and Malik, J. (2015). Viewpoints and keypoints. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1510–1519.

Vilenkin, N. I. (1978). *Special functions and the theory of group representations*, volume 22. American Mathematical Soc.

Wang, R., Cheng, M., Chen, X., Tang, X., and Hsieh, C.-J. (2021a). Rethinking architecture selection in differentiable nas. *arXiv preprint arXiv:2108.04392*.

Wang, S., Mihajlovic, M., Ma, Q., Geiger, A., and Tang, S. (2021b). Metaavatar: Learning animatable clothed human models from few depth images. *Advances in Neural Information Processing Systems*, **34**, 2810–2822.

Wohlhart, P. and Lepetit, V. (2015). Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3109–3118.

Wohlkinger, W. and Vincze, M. (2011). Ensemble of shape functions for 3d object classification. In *2011 IEEE international conference on robotics and biomimetics*, pages 2987–2992. IEEE.

Wong, J. M., Kee, V., Le, T., *et al.* (2017). Segicp: Integrated deep semantic segmentation and pose estimation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5784–5789. IEEE.

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

Xiang, Y., Mottaghi, R., and Savarese, S. (2014). Beyond pascal: A benchmark for 3d object detection in the wild. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*.

Xiang, Y., Schmidt, T., Narayanan, V., and Fox, D. (2017). Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *arXiv preprint arXiv:1711.00199*.

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 *The IEEE International Conference on Computer Vision (ICCV)*.

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*, **2**.

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, A., Ye, V., Tancik, M., and Kanazawa, A. (2021). pixelnerf: Neural radiance fields from one or few images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4578–4587.

Zeitlhöfler, J. (2019). Nominal and observation-based attitude realization for precise orbit determination of the jason satellites.

Zhang, C., Ren, M., and Urtasun, R. (2018). Graph hypernetworks for neural architecture search. *arXiv preprint arXiv:1810.05749*.

Zhang, Z. (1994). Iterative point matching for registration of free-form curves and surfaces. *International journal of computer vision*, **13**(2), 119–152.

Zhou, X., Zhuo, J., and Krahenbuhl, P. (2019a). 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, Y., Barnes, C., Lu, J., Yang, J., and Li, H. (2019b). On the continuity of rotation representations in neural networks. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5745–5753.