Symmetry-Induced Ambiguity in Orientation Estimation from RGB Images

Bertens, T.E.W.

Award date:
2022

Department of Mathematics and Computer Science
Visualization Research Group

Symmetry-Induced Ambiguity in Orientation Estimation from RGB Images

Master's Thesis

T.E.W. Bertens

Supervisors:
dr. A.C. Jalba
dr. A. Saccon

Coach:
B.J. Caasenbrood, MSc.

Members of the assessment committee:
dr. A.C. Jalba
dr. A. Saccon
dr. V. Menkovski

Eindhoven, July 2022

Acknowledgements

With this thesis, I conclude my studies at Eindhoven University of Technology. My thanks go out to my supervisors Andrei Jalba and Alessandro Saccon, whose exceptional guidance has been invaluable during my work on this project. In our frequent meetings, your enthusiasm, patience, advice, confidence, and insight, have always kept me motivated. I am also profoundly grateful to Brandon Caasenbrood for his valuable feedback and advice. You have truly gone above and beyond to support me in this project and it has been a pleasure to work with you. I wish you well in the final stages of your PhD. Finally, I wish to thank my friends, sister, and parents for their unwavering support. At times, you have had more confidence in me than I had in myself, for which I will always be grateful.

Abstract

The estimation of object orientation from RGB images is a core component in many modern computer vision pipelines. Traditional techniques mostly predict a single orientation per image, learning a one-to-one mapping between images and rotations. However, when objects exhibit rotational symmetries, they can appear identical from multiple viewpoints. This induces ambiguity in the estimation problem, making images map to rotations in a one-to-many fashion. In this thesis, we explore several ways of addressing this problem. In doing so, we specifically consider algorithms that can map an image to a range of multiple rotation estimates, accounting for symmetry-induced ambiguity. Our contributions are threefold. Firstly, we create the first orientation estimation data set with annotated symmetry information that covers symmetries induced through self-occlusion. Secondly, we compare and evaluate various learning strategies for multiple-hypothesis prediction models applied to orientation estimation. Finally, we propose to model orientation estimation as a binary classification problem. To this end, based on existing work from the field of shape reconstruction, we design a neural network that can be sampled to reconstruct the full range of ambiguous rotations for a given image. Quantitative evaluation on our annotated data set demonstrates its performance and motivates our design choices.

Contents

Chapter 1

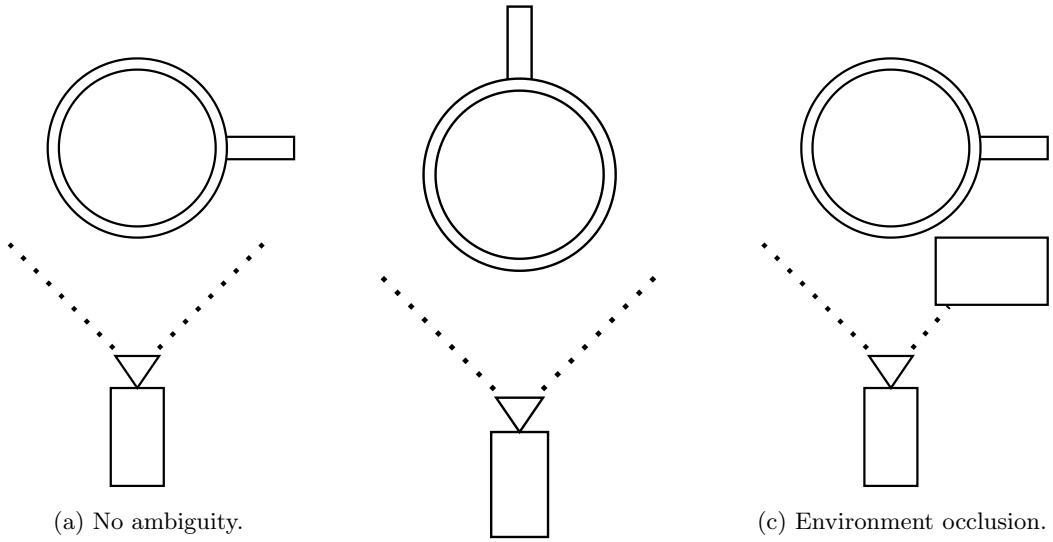
Introduction

1.1 Context and Motivation

Progress in the field of computer vision has had a profound impact on the capabilities of modern robotic systems. Early robotic systems mostly operated in heavily controlled and static environments, performing series of rigidly defined movements and actions, that could be pre-programmed. However, as industry and society push to automate increasingly complex tasks, the capabilities of autonomous systems must increase accordingly. In some tasks, the environment cannot be controlled, and objects can appear in uncertain configurations. Examples of such tasks include autonomous driving [1], collaborating with humans [2], and robotic grasping of objects in dynamic environments [3, 4]. For a robotic system to operate in such an environment, it requires the capability to perceive the dynamic physical world around it. To this end, robotic arms, autonomous cars, and other machinery, are commonly equipped with a variety of sensors. In this thesis, we focus on one specific type of sensor, namely: a camera capturing RGB images. Through the use of vision, images can give rich information about the state of the world surrounding a robotic system. Algorithms to retrieve this information are studied in the field of computer vision.

A core component in many modern computer vision pipelines is pose estimation. Pose estimation is the process of predicting the position and orientation (pose) of objects in images, relative to the camera. This information allows a robotic system to build an internal map of its surroundings, capturing where objects are in relation to itself. In turn, this allows the system to approach, grab, avoid, or otherwise interact with objects in its environment. Though the performance of recent works on pose estimation has been impressive, there are still a number of key challenges to overcome [5, 6]. One such challenge, which has seen an increased amount of attention in recent years, is the influence of ambiguity. More specifically, some objects may appear visually identical from different viewpoints, causing images to map to poses in a one-to-many fashion. This phenomenon mainly occurs as a consequence of symmetries that can be present in the observed object, or be induced by self occlusion or occlusions caused by the environment (Figure 1.1).

Traditionally, pose estimation techniques have been designed to predict only a single output pose. As such, these methods fail to report the multitude of possibly correct poses for symmetric objects. Moreover, when approached naively, the presence of ambiguities can be detrimental to the performance of single-output techniques. Deep neural networks, which are commonplace in pose



(a) No ambiguity. (b) Self occlusion. (c) Environment occlusion.

Figure 1.1: A mug is not symmetric, but occlusions can still induce symmetries. If the ear of the mug is visible, then there is no ambiguity (a). However, if the ear is occluded, the mug appears like a symmetric cylinder (b-c).

estimation, are known to minimize the penalty between their single prediction and all ambiguous poses appearing in the data set. As a consequence, such networks will converge to a conditional average pose, which renders them often meaningless for this situation.

1.2 Recent Progress

Recent works have addressed symmetry-induced ambiguity to various degrees. A first category of works aims to prevent symmetries from disturbing the training process of neural networks. One approach is to apply canonicalization of symmetric poses, mapping all symmetric poses to a single canonical pose [7, 8, 9]. However, this approach requires upfront knowledge about the exact symmetries present in objects, which becomes infeasible when considering the complex ambiguities caused by occlusions. A different approach is to apply symmetry-aware loss functions, like the ShapeMatch-Loss [10], which uses a polygonal mesh to identify symmetric poses automatically. Nevertheless, this loss cannot deal with symmetry-breaking textures or induced symmetries. Furthermore, since only a single pose is predicted, no information about possible ambiguities can be deduced from the output, which would be valuable in downstream tasks like robotic grasping and next-best view prediction.

A more recent, second category of works addresses symmetry-induced ambiguity indirectly as a cause of uncertainty. When considering a probability distribution over the possible poses for an observed object, symmetric poses appear as equally probable outputs. Several recent works have used Bingham distributions [11, 12, 13], Von Mises distributions [14], and Matrix Fisher distributions [15] to output such distributions. While these methods are indirectly able to predict multiple poses, their distributions require computationally expensive normalization and complex parameterizations. Moreover, the choice of distributions limits the expressivity of the algorithm.

Non-parametric distributions can address this issue [16, 17], but often requires discretization over the space of poses.

As part of the second category, we highlight the work by Murphy et al. [18], which particularly interesting. To elaborate, inspired by recent advances in implicit shape reconstruction [19], Murphy et al. [18] proposed to encode a probability density function in a shallow fully connected neural network. In doing so, they provide a non-parametric output density that can take arbitrarily complex shapes, and does not require discretization of the space of poses. Sampling from the distribution, however, still requires the brute force computation of a normalization constant by means of sampling a grid of poses.

1.3 Research Goal and Contributions

Though the recent progress mentioned above addresses symmetry-induced ambiguity to an extent, we identify room for improvement. In this thesis, we explore different ways of performing pose estimation on symmetric objects. Given a single RGB image, our goal is to predict not just a single correct pose, but rather to predict a range of multiple possibly correct poses, when considering symmetry-induced ambiguity. However, we note that rotational symmetries are the most common source of ambiguity in pose estimation. As rotational symmetries only cause ambiguity in the orientation of objects, we restrict our work to only estimating orientation. Therefore, we do not perform full pose estimation, but rather focus on the subproblem of orientation estimation. Finally, no prior knowledge on the types of symmetries present in observed objects are assumed to be available, and methods should be able to work with inherent symmetries as well as induced symmetries.

Towards the aforementioned goal, we present the following three contributions in this thesis.

1. We provide new annotations for part of the recently introduced SYMSOL II data set [18]. The data set is recreated, but for every image we provide the complete set of possibly correct rotations, considering rotational symmetries. In doing so, we create the first orientation estimation data set with annotated symmetry information, covering symmetries induced through self-occlusion.

2. Inspired by the recent work of Manhardt et al. [20], we compare and evaluate various training strategies for multiple-hypothesis prediction models. When applied to orientation estimation, we show how these strategies help to produce sets of diverse orientation hypotheses, which can accurately identify multiple symmetric poses for a given image.

3. Finally, inspired by Murphy et al. [18], we propose to model orientation estimation as a binary classification problem. Rather than predicting rotations directly, or predicting a normalized distribution over rotations, our model instead predicts whether an object in a given rotation could have resulted in a given image. The model can be sampled to reconstruct the full range of ambiguous rotations for a given image. Predictions can be interpreted directly, and do not require expensive normalization.

1.4 Report Structure

The rest of this thesis is structured as follows. First, Chapter 2 covers preliminary concepts, providing background for the matter discussed in the rest of the thesis. Then, an extensive literature study is presented in Chapter 3, which categorizes and details relevant recent works. Chapter 4 gives background on multiple-hypothesis prediction models, and details how they are applied to orientation estimation. Subsequently, various training strategies for multiple-hypothesis prediction models are evaluated and compared in Chapter 5. Furthermore, the same chapter details our recreation of the SYMSOL II data set with annotated symmetry information for every image. Chapter 6 explains our newly proposed model, in which we model orientation estimation as a binary classification problem. A quantitative evaluation of the new model is presented in Chapter 7. Finally, conclusions, as well as recommendations for future work, are given in Chapter 8.

Chapter 2

Preliminaries

This section presents preliminary concepts that are fundamental to the matter discussed in this thesis. Firstly, an introductory explanation of the mathematics behind rotations, along with an overview of multiple rotation representations is given in Section 2.1. Then, the basics of modern deep learning for function approximation and feature extraction from images are explained in Section 2.2. Finally, a formalization of the pose estimation problem is given in Section 2.3.

2.1 Rotations

As the primary subject of this thesis concerns the prediction of rotations, it is important to understand what a rotation is. Specifically in three dimensions, this is not as trivial as one might imagine. The extensive body of works studying the mathematics behind rotations is a testament to this. The mathematical constructions with which rotations are modeled, quickly become abstract and complex. In the light of this complexity, we aim to give the reader a surface level understanding, which should suffice to understand the work presented in later chapters. More specifically, the goal of this section is to give the reader a basic and intuitive understanding of the space of rotations $SO(3)$, and how individual rotations in this space can be represented numerically. Interested readers are referred to [21, 22, 23] for further reading.

2.1.1 Rotation Matrices

The properties of rotations around the origin in three-dimensional Euclidean space $\mathbb{R}^3$ are represented in the *special orthogonal group* $SO(3)$. Since rotations are linear transformations on vectors in $\mathbb{R}^3$, they are most intuitively represented as 3×3 matrices. Therefore, $SO(3)$ can be defined as

$$SO(3) = \left\{ R \in \mathbb{R}^{3\times 3} : R^\intercal R = I, \det(R) = 1 \right\} \ , \tag{2.1}$$

where the constraint $\det(R) = 1$ enforces the transformation R to be orientation-preserving, and the constraint $R^\intercal R = I$ implies that the matrix R is orthogonal and length-preserving. Hence, the column vectors of any $R \in SO(3)$ are unit length, and together form an *orthonormal* basis. Moreover, these constraints restrict $SO(3)$ to a subset of $\mathbb{R}^{3\times 3}$, creating a *manifold* embedded in $\mathbb{R}^{3\times 3}$. Finally, rotation matrices form a group, where the group action of composition is represented

by matrix multiplication, as the multiplication of two orthonormal matrices is itself an orthonormal matrix. A rotation matrix is trivially applied to a vector $x \in \mathbb{R}^3$ by performing matrix-vector multiplication $x' = Rx$.

2.1.2 Euler Angles

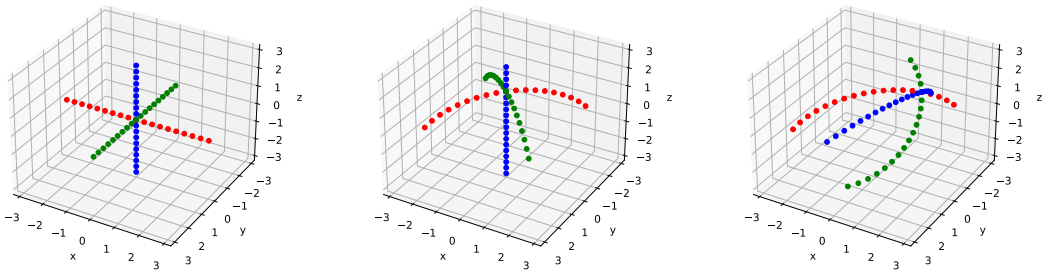
Though matrices are an intuitive way to represent rotations, they are not quite as intuitive for humans to interact with. When thinking about a rotation, it can be difficult to pair a rotation matrix R to that particular rotation. Furthermore, they are rather space-inefficient, as they use nine numbers to represent rotations that have only three degrees of freedom. Because of this, applications in computer graphics and animation often use *Euler angles* to represent rotations that are specified by human users. Euler angles represent rotations in three dimensions as the composition of three separate rotations around the standard basis vectors of $\mathbb{R}^3$. Each such rotation around an axis can be described by a single angle, thus each full rotation in $SO(3)$ is parameterized by a vector (x_1, x_2, x_3), where $x_i \in [0, 2\pi)$.

Although Euler angles are a compact representation, they do not accurately represent the properties of the group $SO(3)$. This can cause problems when working with Euler angles, the most famous of which is *gimball lock* [24]. Other problems are found when performing interpolation or sampling in the space of Euler angles, where a uniform grid of Euler angles does not correspond to a uniformly spaced grid on the manifold $SO(3)$, and the shortest path between a pair of Euler angles similarly cannot easily be related to a distance (or metric in this case) between two rotations on the space of $SO(3)$ [23]. Finally, rotations expressed as Euler angles are hard to compose, usually requiring the angles to be converted to matrices and applying matrix multiplication instead.

2.1.3 Axis-angle Representation

Another useful way of representing rotations is as an axis-angle pair. Following *Euler's rotation theorem*, every three-dimensional rotation can be represented as a rotation around a single axis [23]. Therefore, any rotation can be described by a pair (n, α), where n is a unit vector in $\mathbb{R}^3$ and $\alpha \in [0, 2\pi)$, which describes a rotation by α radians around the axis n. Alternatively, an angle-axis pair can be represented more compactly as a *rotation vector*. A rotation vector takes the angle α and encodes it in the magnitude of n. A rotation is then described by a single vector $n' \in \mathbb{R}^3$, with $|n'| \in [0, \pi]$, which represents a rotation of $|n'|$ radians around the axis codirectional with n'.

Note how the angle of rotation is now restricted to a maximum of π radians. This is, however, no restriction in the expressivity of rotation vectors, since a rotation of $-\alpha$ radians around an axis n is equal to a rotation of α radians around the axis $-n$. An important consequence of this is the equivalence between the rotation vectors n' and $-n'$ when $|n'| = \pi$, which makes the mapping from $SO(3)$ to rotation vectors ambiguous. Nevertheless, with this antipodal symmetry taken into account, rotation vectors are particularly useful for visualization, since they lie completely in $\mathbb{R}^3$. As rotation vectors occupy a ball with radius π and its center on the origin, a plot of rotation vectors is called a *π-ball plot*. Figure 2.1 shows several examples. Note how rotations around the same axis, with respect to the idenitity rotation, appear on a straight line 2.1a. In contrast, rotations around the same axis with respect to a different reference frame can occur as arcs (2.1c).



(a) (b) (c)

Figure 2.1: Examples of π-ball plots showing rotation vectors. Each plot shows 20 rotations around the x-, y-, and z-axis of a different local reference frame. For each axis, the 20 rotations have their rotation angles equally spaced from 0 to 2π radians. Rotations around the x-, y-, and z-axis are colored red, green, and blue respectively. In plot (a), all rotations are relative to the identity rotation, which lies at the origin. In plot (b), the local reference frame is rotated by $\frac{\pi}{2}$ radians around the z-axis relative to the identity. In plot (c), the local reference frame is further rotated by $\frac{\pi}{2}$ radians around the x-axis relative to (b).

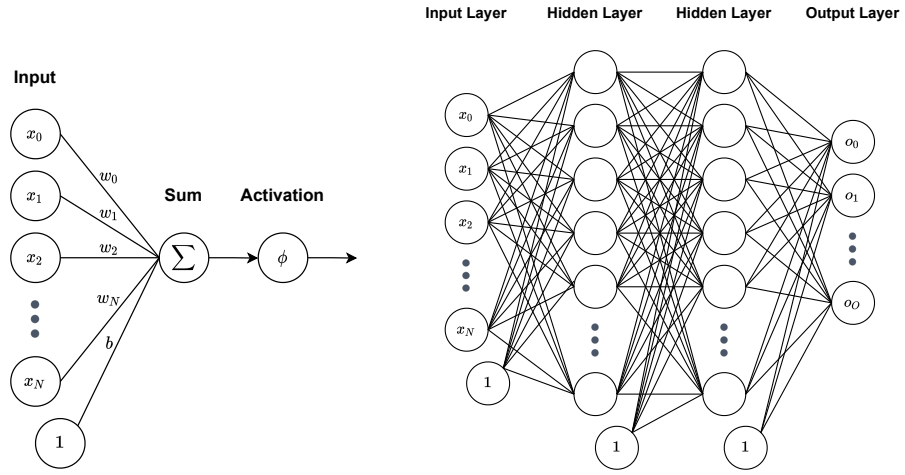
2.1.4 Unit Quaternions

Finally, one of the most commonly used representations are *unit quaternions*. A quaternion is a four-dimensional complex number $q = (q_w, q_x, q_y, q_z) = q_w + q_x i + q_y j + q_z k$, with $i^2 = j^2 = k^2 = ijk = -1$. Unit quaternions have $|q| = 1$ and lie on the hypersphere S^3 embedded in $\mathbb{R}^4$, where all quaternions live. A rotation of α radians around the unit axis $n \in \mathbb{R}^3$ is encoded into a unit quaternion as $q = \cos(\alpha/2) + n\sin(\alpha/2)$, where $n = in_x + jn_y + kn_z$. Note that a rotation of α radians around the axis n is equivalent to a rotation of $-\alpha$ radians around the axis $-n$. Hence, the unit quaternions q and $-q$ represent the same rotation, making them antipodally symmetric. The space of unit quaternions is said to be a *double cover* of $SO(3)$ [23], as every rotation matrix can be represented by exactly two unit quaternions.

Unit quaternions have favorable properties. Firstly, they form a group under the action of the quaternion product. Hence, rotations can be easily composed by performing the quaternion product on their respective quaternion representations [22, 24]. Secondly, the manifold S^3 on which the unit quaternions lie properly represents the topology of $SO(3)$ when taking into account the antipodal symmetry [23].

2.2 Deep Learning

The use of *artificial neural networks* has had a profound impact on the field of computer vision. Manual algorithms for extracting features from images have been replaced by neural networks that learn to extract features based on large collections of training data. Their application in this thesis is almost inevitable considering the rapid advances in neural networks tailored towards vision. However, it is important to underline the word *application* in this sentence. This thesis does not aim to develop new insights in the field of deep learning itself, but rather applies existing work in the field of deep learning to the problem of orientation estimation with symmetry-induced ambiguity. To this end, it is important for the reader to understand the basics of fully connected



(a) A perceptron. (b) A multi-layer perceptron.

Figure 2.2: Schematic diagrams of a perceptron and a multi-layer perceptron.

and convolutional networks, and how they are trained. Here, we briefly detail each of the aforementioned topics, based on the book by Goodfellow et al. [25] on deep learning. For a more extensive background, the reader is referred to this book.

2.2.1 Fully Connected Networks

Perhaps the most popular and fundamental deep learning model is the so-called *multi-layer perceptron* (MLP). An MLP is a general mechanism for approximating functions. More specifically, say there is some function f^* that we wish to approximate, then an MLP can be described as a function f, which can be trained to approximate f^*. To understand how an MLP works, it is important to understand its building block: the *perceptron*.

The design of a perceptron is based on that of neurons found in the brain. Hence, a perceptron is often called an *artificial neuron*, giving rise to the common term *neural networks*. A single perceptron takes a number of inputs organized in a vector $x \in \mathbb{R}^N$, which we assume to lie in N-dimensional Euclidean space for simplicity. The perceptron first applies a linear transformation to the input vector x. To this end, it has a weight vector $w \in \mathbb{R}^N$ and a scalar bias $b \in \mathbb{R}$. Together, w and b form the parameters of the perceptron, and are often collocated into a parameter vector θ. The result of this linear transformation is then passed through an *activation function* ϕ, which is often non-linear. A diagram is given in Figure 2.2a. The perceptron can now be written as a function $f_\theta : \mathbb{R}^N \to \mathbb{R}$, parameterized by θ, such that

$$f_\theta(x) = \phi\left(w^\intercal x + b\right) \ . \tag{2.2}$$

The expressive power of a single perceptron is rather limited. It applies a single linear transformation which is passed through an, optionally non-linear, activation function. To address this, multiple perceptrons with non-linear activation functions can be combined to create a more complex function. Multiple perceptrons can be stacked side-by-side to create a *layer* of perceptrons. Layers can again be composed to create a multi-layer perceptron. This way, subsequent layers

take the output of previous layers as their inputs, creating an increasingly complex mapping from the original input to the final output (Figure 2.2b). The final layer in an MLP is called the *output layer*, the set of inputs is called the *input layer*, and the middle layers are referred to as *hidden layers*. Note that the output layer can contain multiple neurons, thus allowing the output of an MLP to be a vector $o \in \mathbb{R}^O$ in O-dimensional space, where the constant O denotes the number of outputs. Layers of perceptrons are often referred to as *fully connected layers*, since each neuron in a layer is connected to the output of all neurons in the previous layer. Finally, the *width* of a layer denotes the number of neurons in that layer, and the *depth* of a neural network refers to the number of layers in that network. Each individual perception in the MLP has its own weight vector and bias. Therefore, the entire MLP is parameterized by a large vector containing all weight vectors and all biases. Again, this vector is usually called θ.

2.2.2 Training

As already mentioned, perceptrons and MLPs can be used to approximate a function f^*. This implies finding a set of parameters θ such that f_θ best approximates f^*. A way to formalize what the 'best' approximation is to introduce a scoring function $\mathcal{L}$, more commonly called a *loss function*. Conceptually, the loss represents some difference between the ground truth results produced by f^* and the approximated results produced by f_θ. The lower the difference, and therefore the loss, the better the approximation. A simple example would be the popular *mean squared error* (MSE), which, given two vectors $v, v' \in \mathbb{R}^O$, is defined as

$$\mathcal{L}_{MSE}(v, v') = \frac{1}{O} \sum_{i=1}^{O} (v_i - v'_i) \ , \tag{2.3}$$

where subscript i indicates the i^{th} element of a vector. Note how the MSE produces scalar output, making the loss easily interpretable as a single number. Moreover, the MSE is minimized when $v = v'$. Now, given a loss function $\mathcal{L}$, the optimal vector of parameters θ'_{OPT} for which f_θ best approximates f^*, is given by

$$\theta'_{OPT} = \arg\min_\theta \int \mathcal{L}(f_\theta(x), f^*(x))\, dx \ . \tag{2.4}$$

In many practical scenarios, however, the function f^* may not be known. Instead, the only data that is available consists of pairs of examples (x, y) such that $y = f^*(x)$. We call the collection of available pairs the *data set* and define it as $\mathcal{D} = \{(x, y) : x \in \mathbb{R}^N, y = f^*(x)\}$. In this setting, θ'_{OPT} can be approximated by θ_{OPT}, defined as

$$\theta_{OPT} = \arg\min_\theta \sum_{(x,y)\in\mathcal{D}} \mathcal{L}(f_\theta(x), y) \ . \tag{2.5}$$

Usually, however, the data set is split into a *training set* and a *test set*. The training set is, as the name suggests, used to train the model. The test set is used to check its performance on inputs unseen during the training process.

Finding θ_{OPT} is a challenging process, especially when a model contains many parameters. So difficult, in fact, that it is practically infeasible. Nevertheless, the *stochastic gradient descent*

algorithm can be used to find an assignment of parameters that produces a close fit. This algorithm works by iteratively moving the parameters θ in a direction that decreases the total loss produced by the model. To this end, the loss function is expressed as a function of θ, and the gradient of the total loss with respect to θ is computed as

$$\nabla_\theta \sum_{(x,y)\in\mathcal{D}} \mathcal{L}\left(f_\theta(x), y\right) \ , \tag{2.6}$$

where ∇ represents the gradient function. Note that the loss function $\mathcal{L}$ has to be differentiable such that the gradient can be found. When considering θ to be a vector of all parameters in the model, this gradient gives the direction in the space of parameters, into which the loss increases most steeply. Naturally, this implies that moving in the opposite direction, decreases the loss most steeply. Therefore, θ can be updated by iteratively applying the update

$$\theta \leftarrow \theta - \alpha\nabla_\theta \sum_{(x,y)\in\mathcal{D}} \mathcal{L}\left(f_\theta(x), y\right) \ , \tag{2.7}$$

where the scalar α determines the step size, more commonly referred to as the *learning rate* in deep learning. Since the data set can be very large, however, it is computationally very expensive to compute the derivative over the entire data set for every step. Instead, the data set is commonly divided into smaller *batches*. Derivatives are then computed over the total loss for each batch, and a step is taken accordingly. Together, the steps taken by going through all the batches in a data set once make up a single *epoch*.

2.2.3 Convolutional Networks

Images are a special kind of input data for neural networks. Compared to the one-dimensional vectors that are input to an MLP, images are two-dimensional. Since images often come with red, green, and blue channels (RGB), they can even become three-dimensional, as each channel is encoded in a separate two-dimensional image. Moreover, images are considered to be *spatially distributed data*, meaning that features manifest as patterns in neighbouring pixels, and could appear anywhere in the image. For example, when detecting lines, one has to look for pixels that have high contrast with respect to their neighbour, and this pattern could appear anywhere in the image. Hence, in order to properly detect features in images, a mechanism is required that respects the structure of the image data, and detects patterns no matter where they appear in the image.

An example of such a mechanism is a *convolution*. A single convolution is a small filter that is applied to a patch of neighbouring pixels. The convolutions acts as a sort of sliding window that is slid along the image. At each position, the convolution applies a set weights, called a *kernel*, and outputs its activation. Analogous to a perceptron, a single convolution can be modeled as a *neuron*, applying a linear transformation and a non-linear activation function. For example, a typical two-dimensional 3×3 kernel, which could be applied to a two-dimensional image, is parameterized by a weight vector $w \in \mathbb{R}^{3\times3}$ and bias $b \in \mathbb{R}$. Consider a patch of pixels $p \in \mathbb{R}^{3\times3}$,

then the convolution outputs the value

$$\phi\left(\sum_{i=1}^{3}\sum_{j=1}^{3}(w_{i,j}p_{i,j}) + b\right) . \tag{2.8}$$

Applying a convolution to an entire image yields a *feature map* containing its activation at each location in the image (Figure 2.3). Note that every pixel in the feature map is the result of only those pixels in the image that were in the 'field of view' of the convolution. Hence, we say the connection between the two is *sparse*, in contrast to the fully connected perceptron, which applies its weights to all input data at once (Equation 2.2).

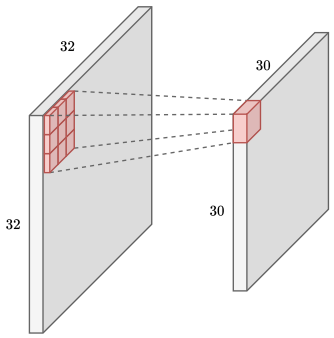
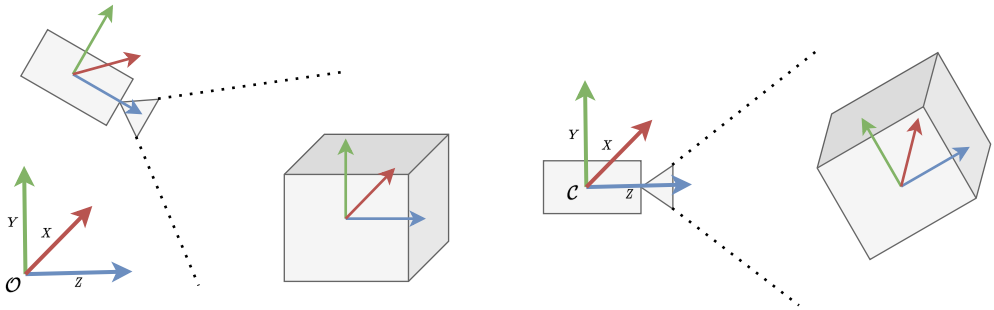


Figure 2.3: A diagram of a convolution being applied to a 32×32 image. Note how the output feature map has size 30×30. This is because there is not enough data around pixels on the edges to fill all pixels of the convolution. Here, only valid positions are considered, called *valid padding*.

Multiple convolutional neurons can again be stacked to create convolutional layers. These layers can again be composed in *convolutional network network* (CNN) to detect increasingly complex features. In between such convolutional layers, it is desirable to progressively decrease the size of feature maps. *Pooling layers* are added to this end, which 'summarize' a set of neighbouring pixels into a single value. Finally, in order to process the features detected by a CNN, a fully-connected network is often added to the end. The final feature map is *flattened* to a one-dimensional vector, and used as input for the fully-connected model. In this context, the CNN is often referred to as a *feature extractor*, *embedder*, or *backbone*, and the fully-connected network as the *head*.

2.3 Pose Estimation

In order for robotic systems to interact with their environment, they require information about the posture of objects in relation to themselves. A common way to retrieve such information is through the use of a camera. By capturing images of the world surrounding the system, objects of interest can be detected and localized. In the most basic setting, each pixel in these images consists of a red, green, and blue color value (RGB), but more advanced cameras are also able to capture depth (RGB-D).



(a) A global reference frame.

(b) The same world in a camera-local reference frame.

Figure 2.4: The position and orientation of objects and the camera are defined relative to a reference frame $\mathcal{O}$ (a). For convenience, object poses are predicted relative to the camera $\mathcal{C}$ (b).

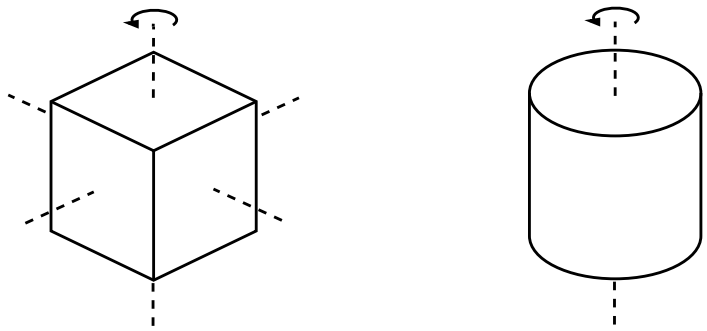
2.3.1 6D Pose Estimation

Objects, as well as the camera itself, appear in the world at different *positions* and in different *orientations*. Assuming the world to be three-dimensional, position can be represented by a vector $t \in \mathbb{R}^3$, representing a *translation* relative to some reference point in the world. Moreover, orientation can be represented by a rotation matrix $R \in SO(3)$, representing a rotation with respect to some reference orientation specific to the object. Together, the translation and rotation have six degrees of freedom, and are referred to as the *6D pose* of an object. More formally, the 6D pose of object is part of $SE(3) \cong \mathbb{R}^3 \times SO(3)$, which is the group of all rigid-body transformations about the origin of $\mathbb{R}^3$. Note that, in this parameterization, objects are assumed to be *rigid*, meaning they do not deform. Furthermore, they are assumed to a have a fixed *scale*, only ever appearing with the same size.

The problem of *pose estimation* is to estimate the pose of an object, given some image captured of that object. Using the aforementioned parameterization of a pose, the problem is also referred to as 6D pose estimation. Since both position and orientation are only defined in relation to a reference frame, a point of reference must be established. Therefore, it is common to define poses relative to the camera (Figure 2.4). In this thesis, we follow the same convention. For more information on the concepts mentioned above, as well as scene reconstruction in general, the reader is referred to the book by Szeliski et al. [22] on computer vision.

2.3.2 Symmetries

One of the main complexities in pose estimation is *symmetry*. In practise, many objects exhibit rotational symmetries, making them appear identical from different viewpoints. Several types of rotational symmetries can be distinguished. Firstly, symmetries can be *discrete* or *continuous*. A cube, for example, has a discrete number of symmetric orientations (Figure 2.5a), while a cylinder has a continuous symmetry about its central axis (Figure 2.5b). Moreover, we distinguish between *symmetric objects* and *nearly symmetric objects*. Symmetric objects have *global symmetries*, that are inherent in the object. In contrast, nearly symmetric objects have a symmetry breaking feature, meaning they have no inherent symmetries. Nevertheless, if the symmetry breaking feature becomes occluded, a symmetry can be induced (Figure 1.1). In pose estimation, these



(a) (b)

Figure 2.5: A cube (a) has a discrete number of symmetric orientations. In contrast, a cylinder (b) is symmetric under a continuous rotation about its central axis. Axes of symmetry are shown in dashed lines.

symmetries result in *ambiguity* on the orientation, as multiple rotations can result in the same observed image. Therefore, in the presence of symmetries, images can map to poses in a one-to-many fashion.

This mapping is formalized in what are called *symmetry sets* [20]. For an image showing an object of interest, there exists a set of poses, such that the shape and appearance of the object projected on the image, are indistinguishable between any of the poses. This set of poses is referred to as the symmetry set of the image. Let $\mathcal{I}$ be the space of all images, then the symmetry set of an image $I \in \mathcal{I}$ is defined as

$$S_{SE(3)}(I) = \{(t, R) \in SE(3) : \mathcal{R}(t, R) = I\} \ . \tag{2.9}$$

Here, $\mathcal{R}$ is defined as an abstract 'rendering' function $\mathcal{R} : SE(3) \to \mathcal{I}$, which can take a 6D pose and produce an image. Many details are abstracted in this function, such as the geometry of the rendered object, lighting conditions, and other environment variables. Importantly, we assume that the object of interest is visible in I. If it were not visible, an $S(I)$ would contain an infinite number poses, all of which transform the object outside the view of the camera.

Dealing with the ambiguity caused by these rotational symmetries is the prime focus of this thesis. Since rotational symmetries only pertain to orientation, we restrict the pose estimation problem to orientation estimation in this thesis. More specifically, instead estimating full 6D poses, we assume the translation is fixed, and only predict orientation. In this context, the definition of a symmetry set, to be used in the rest of this thesis, is changed accordingly to

$$S(I) = \{(R) \in SO(3) : \mathcal{R}(R) = I\} \ , \tag{2.10}$$

where the rendering function is defined only over the rotation manifold $\mathcal{R} : SO(3) \to \mathcal{I}$.

2.3.3 Uncertainty and Ambiguity

Though there are many different ways of performing pose estimation, most modern methods rely on deep learning. Large data sets of images with annotated object poses are used to train neural

networks. When considering the robustness of a neural network, *uncertainty* is a commonly used term. When a network produces output, it is desirable for that output to be accompanied by some measure of certainty, such that the quality of the output can interpreted before it is put to use. For example, if an input is given to a network, which is completely unlike any of the inputs used during training, the network may produce incorrect output. In pose estimation, this can occur when, for example, an object appears under different lighting conditions in the real world compared to the training data. In this case, it is desirable for the network to present its output with low confidence.

Symmetries in pose estimation is often addressed as a source of uncertainty. When an image can map to multiple different poses, any individual pose should be output with low confidence. After all, the network is uncertain as to which of the poses in the symmetry set is correct. However, this interpretation only holds for models that output a single pose. In contrast, if a model properly represents the one-to-many mapping between images and poses, symmetries do not have to cause uncertainty. If a given image closely resembles the training set, such a network could output all possible symmetries with high confidence.

To avoid confusion, it is important to understand the difference between *ambiguity* and *uncertainty*. Ambiguity is an inherent property of the problem domain. An image can map to multiple poses, making the problem of pose estimation ambiguous. Uncertainty, on the other hand, is an emergent property of the neural network and its training process. When a network is unable to make a prediction with high confidence, we speak of uncertainty. Ambiguity can be a source of uncertainty if the network is ill-equipped to deal with it. However, if the model is designed with the ambiguity in mind, a high degree of ambiguity can exist without any uncertainty.

Chapter 3

Related Work

Dealing with symmetric objects has been established as an important open problem is pose estimation [5, 6]. As such, it has received an increasing amount of attention in the rich body of recent literature on the topic. There exist numerous characteristics by which these works can be grouped. In this section we distinguish two major categories: *symmetry-aware learning* and *learning with uncertainty*. Both categories represent a fundamentally different way of approaching the problem. Nevertheless, a number of works do not properly fit this categorization and are presented separately.

3.1 Symmetry-Aware Learning

The first category makes an effort to avoid symmetry-induced ambiguities from disturbing the training process. Here, the goal is neither to report on the presence of symmetries, nor to give a range of possible poses under symmetry. Instead, authors look to produce the best possible single-pose output, the quality of which is invariant between symmetric poses.

3.1.1 Canonicalization

Early works in this direction have opted to simply remove ambiguous cases from training data [8, 9, 26]. Both SSD-6D [8] and the work from Wohlhart et al. [26] completely ignore an axis of rotation during training and evaluation for symmetric objects. In doing so, they manually remove ambiguity from the input and output data. Similarly, Rad et al. [9] restrict poses of symmetric objects to angles between $[0, \alpha)$ around the axis of symmetry, where α is the angle of symmetry. Moreover, they note how poses in the extremities of this range still display high visual similarity. The range is, therefore, further divided into two intervals $r_1 = [0, \alpha/2)$ and $r_2 = [\alpha/2, \alpha)$. A separate network is trained to classify whether the rotation of an object falls inside r_1 or r_2 (Figure 3.1). If it falls in r_2, the input image is mirrored, predicted as if it was in r_1, and then manually translated back to r_2. A very similar approach is taken by Pitteri et al. [7], who identify groups of symmetric poses as equivalence classes. An operator is defined that maps each member of an equivalence class to a canonical pose. In practise, this results in a process very similar to [9], where additional regressors have to be trained to address discontinuities created by the mapping.

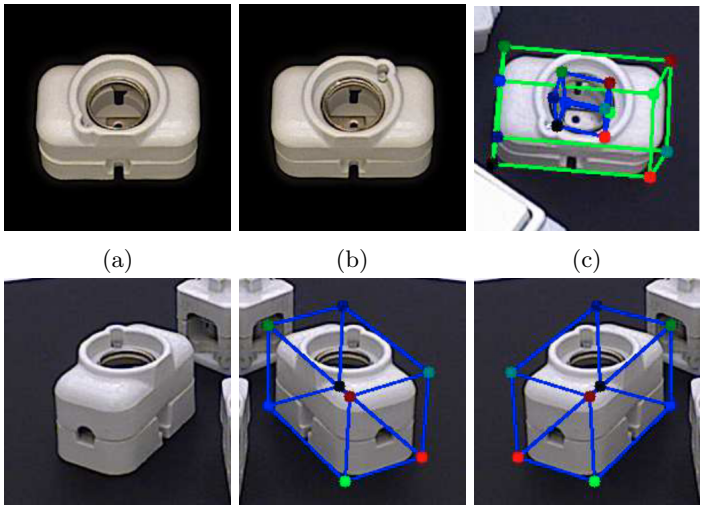


Figure 3.1: An example of how Rad et al. [9] handle symmetric objects. Shown is object #5 from the T-LESS data set, which has an angle of symmetry α of 180 degrees, ignoring the small screw and electrical contact. If the range of poses is restricted in the training set between 0 degrees (a) and 180 degrees (b), pose estimation still fails for test samples with an angle of rotations close to 0 degrees modulo 180 degrees (c). Their solution is to restrict the range during training to be between 0 degrees and 90 degrees. A classifier is used to detect if the pose in an input images is between 90 degrees and 180 degrees. If this is the case (d), the input image is mirrored (e), and the predicted projections for the corners are mirrored back (f). Figure and caption taken from [9].

Note that all the aforementioned works require explicit knowledge of object symmetries during training time. Furthermore, they require manual work to identify axes of symmetry, and implement special cases in data-generation for each symmetric object.

3.1.2 Symmetry-Aware Loss

More recent works have started using loss functions that do not penalize poses symmetric to the 'correct' training pose. In doing so, there is no longer a need to manually curate training data. Park et al. [27] introduced the *transformer loss* in their framework Pix2Pose. For a given training sample, the transformer loss computes the minimum loss between the predicted pose and all poses symmetric to the sample. Note that this approach still requires upfront knowledge, namely in the form of a database of symmetric pose candidates for all objects. Brégier et al. [28] implement a similar concept, but by embedding poses in a higher dimensional space, in which the distance between two poses can be evaluated as their Euclidean distance. The embedding function depends on the type of symmetry present in the object. Poses for objects with discrete symmetries are represented as sets of points in the embedding space, and the distance to such a pose is the minimum with respect to all its embedded points. In contrast, the *ShapeMatch-Loss* introduced in PoseCNN [10] requires no prior knowledge at all. Using a 3D model of the object, it measures the distance between points on the object in its predicted orientation and the *closest* point on the

ground truth object. More formally, given a set of model points $\mathcal{M}$, the loss is defined as

$$SLoss(R_p, R_{GT}) = \frac{1}{2m} \sum_{x_1 \in \mathcal{M}} \min_{x_2 \in M} ||R_p x_1 - R_{GT} x_2||^2 \ , \tag{3.1}$$

where R_p and R_{GT} are the predicted and ground truth rotations respectively, m is the number of points, and $||\cdot||$ computes the length of a vector. In this formulation, symmetric predictions will naturally result in the same loss value, without needing to know explicitly about axis of symmetry in the object.

Notably, works in this category almost exclusively target global object symmetries, ignoring occlusion-induced symmetries. Manually creating a database of all symmetric pose candidates becomes unfeasible when considering view-dependent symmetries. Moreover, by focusing only on the object's geometry, ShapeMatch-Loss fails on nearly-symmetric objects. An occluded symmetry-breaking feature would still have to be predicted correctly for the shape in the predicted pose to match the ground truth. Furthermore, possible symmetry-breaking features in the object's texture are ignored.

3.2 Learning with Uncertainty

A second category of works tackles symmetry-induced ambiguities under the broader concept of uncertainty. The most prominent line of work in this direction aims at outputting probability distributions over poses, rather than predicting poses directly. More specifically, given an image, the conditional distribution over poses given the image is predicted. In this setting, symmetry-induced ambiguity manifests as multiple output poses with equal probability in the conditional distribution. An accurate estimation of this distribution could, therefore, be used to detect and explain symmetries present in an image. We distinguish between methods that output parametric and non-parametric distributions.

3.2.1 Parametric Distributions

One option is to use parametric distributions from directional statistics to estimate the conditional pose distribution. The benefit of having a small set of parameters is that they are easily interpretable and can be estimated by a neural network directly. Multiple such distributions have been used by recent works including the Von Mises distribution [14], Bingham distribution [11, 13, 12], and Matrix Fisher distribution [15]. Both the Bingham and Matrix Fisher distributions require the computation of a normalizing constant during training, requiring expensive interpolation [11, 13] and approximation [15] schemes.

While Mohlin et al. [15] output the parameters of a single Matrix Fisher distribution, others have moved in the direction of outputting mixtures of multiple distributions. The latter allows for estimating multimodal distributions, which are a common occurrence when symmetry-induced ambiguities are involved. Prokudin et al. [14] show how to construct a mixture model with an infinite number of mixing components using a conditional variational autoencoder (CVAE). Their architecture is based on a biternion networks [29], which output 2D rotations in the form of 2D vectors $(\cos(\varphi), \sin(\varphi))$, where φ is an angle of rotation. These vectors are coined biternions in

[29], and correspond to unit quaternions around a fixed reference axis. Hence, the CVAE can only predict 2D poses, and is applied to 2D head pose estimation in the paper. Nevertheless, the authors do create a 3D implementation, where each axis is predicted separately. Deng et al. [13] propose Deep Bingham Networks (DBNs), which outputs parameters for a mixture of Bingham distributions. Mode collapse is prevented by using a 'Winner Takes All' (WTA) strategy during training.

While these parametric distributions form a convenient and interpretable output format for neural networks, their expressiveness is limited. By choosing a specific distribution, or mixture thereof, an assumption is made on the shape of the conditional pose distribution. In the presence of symmetry-induced ambiguity, the conditional distribution is uniform over all symmetric poses and is often multimodal. However, all the aforementioned techniques struggle to accurately represent this uniformity, as they use non-uniform priors.

3.2.2 Non-Parametric Distributions

Non-parametric distributions provide an alternative that does not make assumptions on the shape of the predicted distribution. As such, they have the potential to more accurately represent the complex conditional pose distributions caused by symmetries. We highlight three important works from this category.

Deng et al. [17] use a Rao-Blackwellized particle filter to track the posterior distribution over object poses over time. To this end, the posterior is decomposed into separate translation and rotation components. The space of rotations is discretized into 191 thousand bins and the likelihood of a given discretized pose with respect to a given observation is determined by code book matching. An augmented auto-encoder [30] is used to this end, which embeds observed and pre-rendered images into a shared embedding space, in which similarity scores can be computed. Poor translation estimates also result in poor similarity scores, in turn allowing for the computation of likelihoods for translation hypotheses.

Okorn et al. [16] similarly use code book matching to create a discretized non-parametric histogram distribution over rotations. Rather than using an auto-encoder, they instead learn a comparison function between features of pre-rendered images and observed images. Likelihoods can be computed for each of the discretized rotations by inferring this learned similarity metric. These likelihoods can, furthermore, be interpolated to create a continuous distribution.

Finally, *implicit-pdf* [18] takes a rather different approach. The authors estimate conditional rotation distributions as unnormalized functions over the combined space of images and rotations, which can be parameterized by a neural network. The architecture is divided into two conceptual stages, where the first embeds images into feature vectors using a CNN, and the second concatenates such an embedding with an encoded rotation matrix, which is fed into a shallow fully connected network. The small size of the second stage allows for efficiently evaluating multiple sample rotations for a single image. Using these samples, a normalizing constant can be approximated, which is used to turn the unnormalized output into a proper probability distribution.

Considering all the work mentioned above, implicit-pdf is the only work that does not require discretizing the rotation space. Moreover, the predicted probability distribution can be inferred using the full expressive power of a multi-layer neural network. Hence, it can be almost arbitrarily

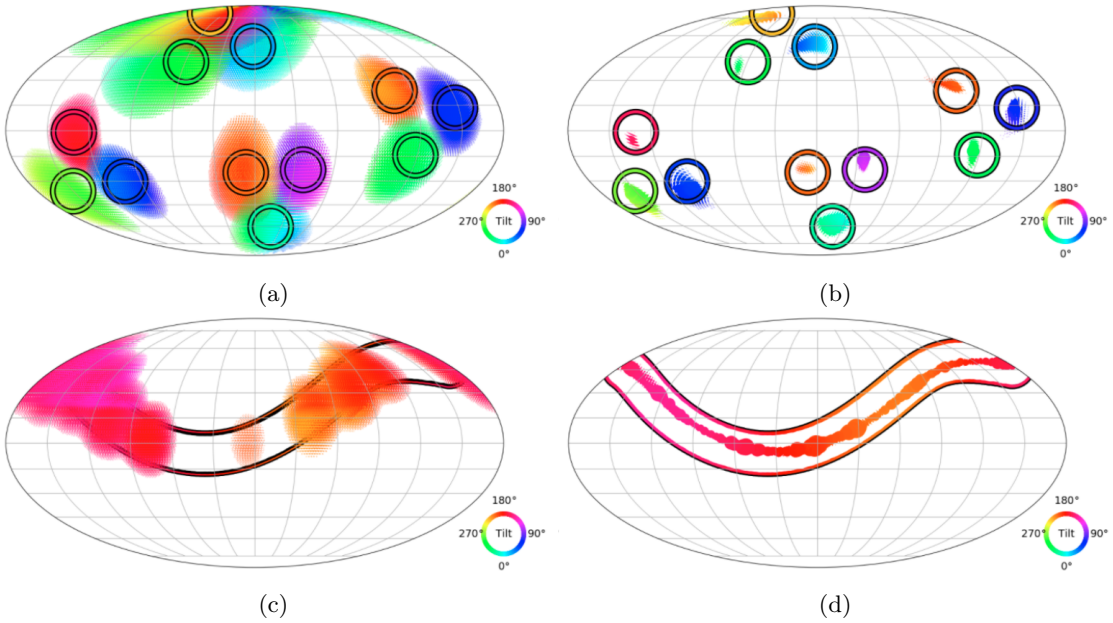


Figure 3.2: A comparison of predicted orientation distributions between Deng et al. [13] (left column) and Murphy et al. [18] (right column), on a tetrahedron (top row) and cone (bottom row). The method by Deng et al. outputs a mixture of Bingham distributions. Though their method is able to identify all 12 symmetries of the tetrahedron (a), the mixture is not expressive enough to represent the continuous symmetry of the cone (c). The predicted distributions by Murphy et al. are much more densely concentrated around the ground truth (b), and can properly represent continuous symmetries (d). Distributions are presented by viewing each point on the 2-sphere as the direction of a canonical z-axis. Color indicates the tilt angle about that axis. The surface of the 2-sphere is visualized using the Mollweide projection. Rotations with larger probability are shown with larger points. Images taken from [18].

complex (Figure 3.2). These desirable features make implicit-pdf one of the primary groundworks for the work presented in Chapter 6 of this thesis.

3.3 Other Works

Given the aforementioned works, there exist a number of recent works do not fit the previously established categories. Nevertheless, they provide valuable alternatives for dealing with pose ambiguity. A number of these techniques are highlighted in the following.

2D-3D Correspondences A popular line of work in pose estimation is to generate 2D-3D correspondences between image pixels and object surface coordinates. These correspondences can subsequently be used in combination with, for example, a PnP-RANSAC algorithm to retrieve a pose. In this category, several works have addressed objects with symmetries. EPOS [31] divides the object surfaces into fragments. An encoder-decoder network is used to predict the probability of an image pixel corresponding to each of the fragments, thus establishing a many-to-many relation between pixels and surface fragments. Symmetries will naturally cause pixels to map to multiple fragments with equal probability. In contrast, Richter-Klug et al. [32] remove

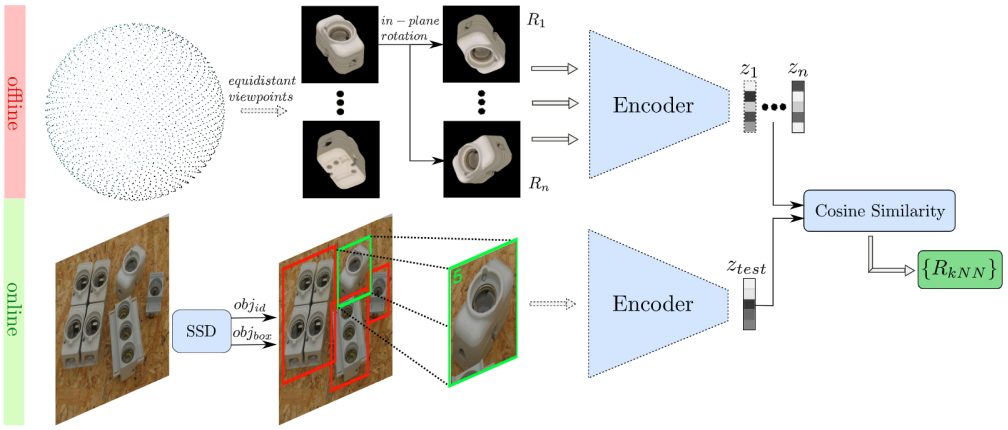


Figure 3.3: Sundermeyer et al. [30] train an encoder network to embed images into a latent space. A code book is created that pairs a number of discretized rotations $R_i \in SO(3)$ to an associated latent code z_i. At test time, objects are cropped from larger images, and encoded into latent vectors z_{test}. The code book entry best matching z_{test} according to cosine similarity is used to predict one or more rotations.

ambiguity from the decoder output space. To this end, they introduce the *closed symmetry loop* (CSL) representation, which maps symmetric object surface points to the same value. A secondary mapping is also construct which maps points from the csl representation back to object surface coordinates. This process is best seen as a form of canonicalization, as described earlier in this Section.

Image Template Matching Another class of pose estimation algorithms focuses on image template matching. An observed image is matched against template images containing objects in known poses. If, by some comparison measure, a match is found, the pose of the observed object can be estimated to be the template's associated pose. This approach has the potential to deal with symmetries naturally, as symmetric objects will create similar template images in symmetric poses. The same holds for nearly-symmetric objects, which will exhibit self-occlusion induced symmetries in template images.

In this category, a common strategy is to embed real images and pre-rendered images with known views into a shared latent space. A library of latent codes for such pre-rendered images is often constructed called a code book. Corona et al. [33] train an encoder network using a triplet loss [26] to predict poses of objects unseen at training time. Notably, they add a separate branch to their network, which also classifies an object's order of rotational symmetry along the X-, Y-, and Z-axis. Sundermeyer et al. [30] use an augmented autoencoder (AAE) to embed observations into a latent space. The closest match in the code book according to cosine similarity is chosen as the output orientation (Figure 3.3). Alternatively, the algorithm can return the K-nearest neighbours of the embedded input, which would give information about other similar poses. If the nearest neighbours are spread out, it implies a high degree of certainty. If, on the other hand, the nearest neighbours are clustered, it implies that a number of poses could be equally likely. This can serve as an indication that the input can be explained by multiple symmetric poses, since such poses will naturally be clustered in the latent space.

Multiple-Hypothesis Prediction Manhardt et al. [20] take a different approach. They propose a model that produces multiple pose hypotheses for a single image. To this end, they train a network with duplicated output layers using a 'Winner Takes All' strategy. The range of predicted hypotheses can be interpreted as a Bingham distribution over possible poses explaining the input. How close the hypotheses lie together, moreover, serves as an indication of the presence of ambiguity. More specifically, if there is no ambiguity, all hypotheses will lie close together. If there is ambiguity, the hypotheses will be spread out. By performing further analyses, the axis of symmetry can even be reconstructed from a set of hypotheses.

Compared to the other works mentioned in this chapter, these multiple-hypothesis prediction (MHP) models come with a desirable set of properties. They neither require complex parameterization, nor the manual canonicalization of poses before or during the training process. Moreover, they are able to directly predict a set of multiple, possibly symmetric, poses. In Chapter 4 of this thesis, we further explore different variations of the winner takes all training strategies, and how they may apply to orientation estimation. Furthermore, in Chapter 5, we evaluate how well hypotheses from MHP models are able to represent the symmetry sets of images.

Chapter 4

Multi-Hypothesis Orientation Estimation

In this Chapter, we cover the concept of multiple-hypothesis prediction in deep learning, and explain how it can be applied to orientation estimation. Section 4.1 first details how multiple-hypothesis prediction models can address ambiguity in learning tasks, as well as introduces multiple training strategies for such models. Subsequently, 4.2 details the application of multiple-hypothesis prediction to orientation estimation, covering two possible loss functions and arriving at a final neural network architecture.

4.1 Background

The problem of ambiguity in orientation estimation boils down to a more fundamental one. As such, we will treat the problem of ambiguity step-by-step. We first focus on the problem of ambiguity in general deep learning. Ambiguity exists, when for a given input x, there can be multiple correct outputs y. Consider a data set $\mathcal{D}$ consisting of N input-output pairs from an input space $\mathcal{X}$ and an output space $\mathcal{Y}$. More formally, we define

$$\mathcal{D} = \{(x_i, y_i) : 1 \leq i \leq N, x_i \in \mathcal{X}, y_i \in \mathcal{Y}\} \ . \tag{4.1}$$

Now, let (x_i, y_i) and (x_j, y_j) be two separate pairs belonging to the data set $\mathcal{D}$, such that $i \neq j$. Then, ambiguity implies that there exist such data pairs, where $x_i = x_j$ and $y_i \neq y_j$. In other words, two different pairs can have the same input, but with different corresponding output.

A traditional neural network can be seen as a function $f_\theta : \mathcal{X} \rightarrow \mathcal{Y}$ with parameters θ. During training, a loss function $\mathcal{L}$ is minimized over the entire data set by minimizing the value

$$\frac{1}{N} \sum_{i=1}^{N} \mathcal{L}\left(f_\theta(x_i), y_i\right) \ . \tag{4.2}$$

Adopting the notation from [34], it is assumed that the above sum of errors, given large enough

N, approximates the continuous formulation

$$\int_{\mathcal{X}} \int_{\mathcal{Y}} \mathcal{L}\left(f_\theta(x), y\right) p(x, y)\, dy\, dx \ , \tag{4.3}$$

where $p(x, y)$ is the joint probability density over x and y. If the network is allowed to produce only a single output, the total loss is minimized when, for a given x, the network outputs a value such that

$$f_\theta(x) = \min_{y_o \in \mathcal{Y}} \int_{\mathcal{Y}} \mathcal{L}(y_o, y) p(x, y) dy \tag{4.4}$$

$$= \min_{y_o \in \mathcal{Y}} \int_{\mathcal{Y}} \mathcal{L}(y_o, y) p(y|x) p(x) dy \tag{4.5}$$

$$= \min_{y_o \in \mathcal{Y}} \int_{\mathcal{Y}} \mathcal{L}(y_o, y) p(y|x) dy \ . \tag{4.6}$$

In other words, for a given input x, the network learns to predict a value that minimizes its loss with respect to all output values weighted by their conditional probability. If the mapping from $\mathcal{X}$ to $\mathcal{Y}$ is one-to-one or many-to-one, meaning there is no ambiguity as all $x \in \mathcal{X}$ map to distinct $y \in \mathcal{Y}$, then it is easy to show how the network regresses to the right answer. Given x, let y_{GT} be the corresponding unambiguous output. Then, by definition, y_{GT} has a conditional probability of 1, while all other values $y \in \mathcal{Y} \setminus y_{GT}$, have conditional probability 0. The optimal prediction is now defined as

$$f_\theta(x) = \min_{y_o \in \mathcal{Y}} \mathcal{L}\left(y_o, y_{GT}\right) \ . \tag{4.7}$$

For any proper loss function $\mathcal{L}$, which is minimized when the prediction is equal to the ground truth, this implies that $f_\theta(x) = y_{GT}$.

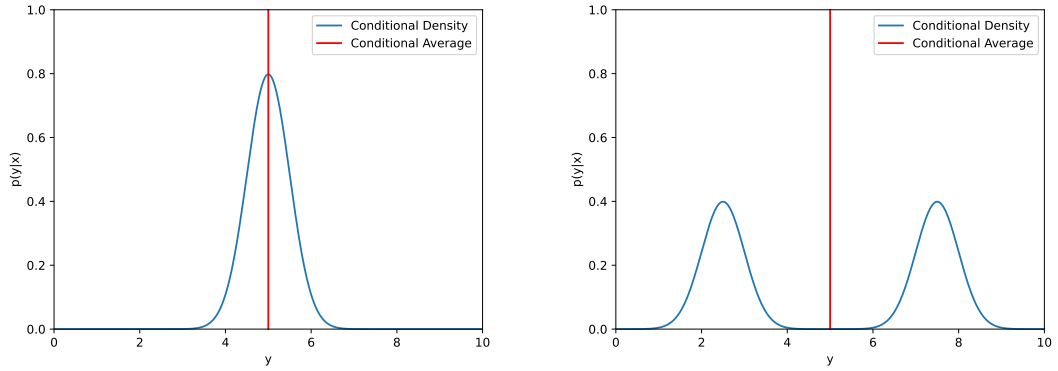
However, if the mapping from $\mathcal{X}$ to $\mathcal{Y}$ is one-to-many, meaning there is ambiguity, then this is not the case. Instead, a given x can map to a variety of values $y \in \mathcal{Y}$. We define $\mathcal{A}(x)$ to be the set of all such values y, for which the pair (x, y) could appear in $\mathcal{D}$. For the sake of consistency in notation, we assume $\mathcal{A}(x)$ is an infinite set, though it could also have a discrete number of elements. The optimal prediction is now defined by integrating over $\mathcal{A}(x)$ as follows

$$f_\theta(x) = \min_{y_o \in \mathcal{Y}} \int_{\mathcal{A}(x)} \mathcal{L}(y_o, y) p(y|x) dy \ . \tag{4.8}$$

As was mentioned before, this implies that the network learns to predict an output that minimizes the loss with respect to all $y \in \mathcal{A}(x)$, weighted by the conditional probabilities $p(y|x)$. Where exactly this optimum prediction lies heavily depends on the loss function $\mathcal{L}$. Take for example the standard mean squared error (MSE) loss, for which the optimum prediction would converge to the conditional average

$$f_\theta(x) = \int_{\mathcal{A}(x)} y \cdot p(y|x)\, dy \ . \tag{4.9}$$

Whether this is a desirable property depends on the shape of the conditional density $p(y|x)$. Consider the densities in Figure 4.1. If $p(y|x)$ is unimodal and symmetric, like in Figure 4.1a, then the conditional average corresponds to the most likely output y. However, if $p(y|x)$ is skewed or multimodal, like in Figure 4.1b, then the conditional average may fall outside the distribution



(a) A unimodal density.

(b) A multimodal density.

Figure 4.1: A visualization of the conditional average for a unimodal distribution and a multimodal distribution.

entirely. The latter case can occur when multiple distinct elements $y_a, y_b \in \mathcal{A}(x)$ exist, with $y_a \neq y_b$, such that both y_a and y_b correspond to a peak in $p(y|x)$. As demonstrated in Equation 4.9, in such a case, the prediction $f_\theta(x)$ will converge to be in between y_a and y_b, thus giving a meaningless value as it represents neither of the most probable outputs.

4.1.1 Multiple Hypothesis Prediction

Now that the problem of training a single-output network to learn on ambiguous data has been established, we introduce the concept Multiple Hypothesis Prediction (MHP) models. The term was first coined by Rupprecht et al. [34] in 2017, but the concept can be traced back to the work by Guzman-Rivera et al. [35] in 2012. The approach was first introduced in the field of multiple choice learning (MCL). Observing that uncertainty and ambiguity can not be properly captured by single-prediction models, Guzman-Rivera et al. proposed to output a *set of multiple plausible hypotheses* instead. As long as one of the hypotheses is correct, they argue that a downstream entity with more knowledge should be able to identify it. Hence the name: multiple *choice* learning. This entity could, for example, be a human that is presented a set of possibilities in an interactive application. Alternatively, it could also be another algorithm that is able to score each hypothesis. One example is the application of human pose estimation, where a set of proposed key-point locations is passed to a downstream algorithm, which will incorporate temporal information to pick the best option. Guzman-Rivera et al. [36] later proposed such an application for the problem of camera relocation. There, they render a depth map from each hypothesized camera location and compare it to the original input depth. The hypothesis with the highest corresponding depth map is chosen as the final prediction. With the assumption that such an oracle exists, works in the line of MCL have focused on producing sets of hypotheses that are as diverse as possible [37]. After all, the more diverse the hypotheses, the more chances to find the right output. In the absence of an oracle, recent works by Lee et al. [38] and Tian et al. [39] have proposed methods to assign a notion of confidence to each individual hypothesis, thus providing an inherent scoring mechanism to an MCL model.

To produce these multiple hypotheses, MCL models traditionally consist of an ensemble of neural networks. A given input x is fed to each of the M networks in the ensemble, producing a set of outputs

$$f_\theta(x) = \left(f^1_{\theta_1}(x), ..., f^M_{\theta_M}(x)\right) \ , \tag{4.10}$$

where each $f^i_{\theta_i}(x)$ is referred to as a single hypothesis. Note that in this formulation each network $f^i_{\theta_i}(x)$ is parameterized by its own set of parameters θ_i, making them completely independent. Later work has suggested that feature sharing between members of the ensemble is beneficial [38, 34]. Rupprecht et al. have even gone so far as to implement the ensemble as a single network, just with M duplicated output layers. In this case, the parameter sets θ_i, $1 \leq i \leq M$, obviously overlap greatly. Hence, it is more convenient to see all hypotheses as different outputs of the same parameterized network:

$$f_\theta(x) = \left(f^1_\theta(x), ..., f^M_\theta(x)\right) \ . \tag{4.11}$$

To reach the aforementioned goal of diversifying the hypotheses, Guzman-Rivera et al. [35] fist introduced the *oracle loss*. The name reflects the fundamental assumption in MCL, namely that there exists an oracle that is always able to identify the best hypothesis. Using the notation of equation 4.11, the oracle loss for a data pair $(x, y) \in \mathcal{D}$ is defined as follows:

$$\mathcal{L}_O(x, y) = \min_{1 \leq i \leq M} \mathcal{L}\left(f^i_\theta(x), y\right) \ , \tag{4.12}$$

where $\mathcal{L}$ is a loss function that can be chosen freely, based on the application domain. In words, the model only pays a loss $\mathcal{L}(f^i_\theta(x), y)$ for the best hypothesis $f^i_\theta(x)$, which is the one creating the smallest task-dependent loss among all hypotheses. Therefore, the oracle loss emulates an oracle choosing the best hypothesis. In doing so, it allows all other hypotheses to lie freely distributed over the output domain without being penalized. In turn, individual networks f^i_θ are allowed to *specialize* in predicting values located in specific regions of the output space.

The 2017 work by Rupprecht et al. focuses precisely on this last observation. Though the proposed multiple hypothesis prediction (MHP) models by Rupprecht et al. are very similar to MCL models, they are used as means to a different end. While MCL is used to pass a set of plausible hypotheses to a downstream oracle, Rupprecht et al. propose MHP models to identify the presence of uncertainty and to discover multiple modes in multimodal distributions. To this end, their analyses focuses on the behaviour of hypotheses with respect to the conditional density $p(y|x)$, when training using the oracle loss. In their analysis, they relate the training process to the mathematical concept of Voronoi tesselations. In this context, they find that the continuous error formulation from Equation 4.3 can be reformulated to

$$\int_\mathcal{X} \sum_{j=1}^{M} \int_{\mathcal{Y}_j(x)} \mathcal{L}\big(f^j_\theta(x), y\big) p(x, y) \, dy \, dx \ , \tag{4.13}$$

where $\mathcal{Y}_j(x)$ represents a subregion of the output space in which the loss between any point $y \in \mathcal{Y}_j(x)$ and the hypotheses is minimal for $f^j_\theta(x)$. These are all the points for which the oracle loss would select f^j_θ. Conceptually, the output space $\mathcal{Y}$ is thus subdivided into a voronoi

tessellation. More formally, the regions are defined as

$$\mathcal{Y}_j(x) = \left\{ y \in \mathcal{Y} : \mathcal{L}\big(f_\theta^j(x), y\big) < \mathcal{L}\left(f_\theta^k(x), y\right) \ \forall k \neq j \right\} \ . \tag{4.14}$$

Assuming a mean squared error loss for $\mathcal{L}$, Rupprecht et al. show that the loss in Equation 4.13 is minimized when the hypotheses tesselate the output space into cells with minimal expected loss to their conditional average.

4.1.2 Training Schemes

In works subsequent to that of Rupprecht et al., the name 'oracle loss', as used in the MCL literature, has been largely replaced by the name 'winner takes all' (WTA) loss. The aforementioned property of creating a minimizing Voronoi tesselation has made the WTA loss popular in multimodal prediction problems. Moreover, the WTA loss works with any application-specific loss function $\mathcal{L}$, making it easily adaptable to almost any problem. Because of this property, WTA is often referred to as a *meta loss function*, which can be used together with an arbitrary *embedded* or *nested* loss function $\mathcal{L}$. We will now give a more detailed description of the WTA training strategy, as well as two adaptations: relaxed winner takes all (RWTA) and evolving winner takes all (EWTA).

WTA

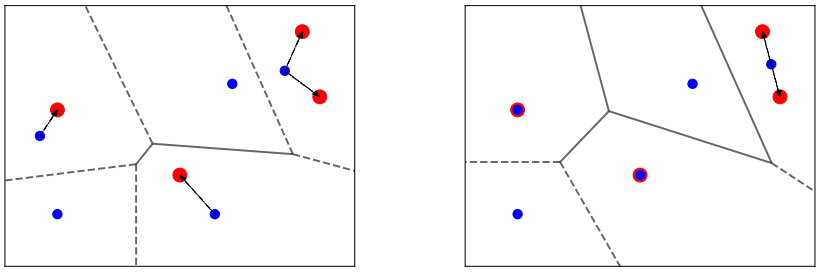
As was mentioned before, the WTA loss is simply the same as the oracle loss, but with a different name. Hence, we define it as

$$\mathcal{L}_{WTA}(x, y) = \min_{1 \leq i \leq M} \mathcal{L}\left(f_\theta^i(x), y\right) \ . \tag{4.15}$$

In order to illustrate how the WTA meta loss function influences the convergence of hypotheses, we consider a simple example. Let $x \in \mathcal{X}$ be an ambiguous input sample, such that the distribution $p(y|x)$ is multimodal. For the sake of simplicity, we assume that there exist a discrete number of four ambiguous outputs for x, hence $|\mathcal{A}(x)| = 4$. For an MHP model with $M = 5$ randomly initialized hypotheses, Figure 4.2 shows how the hypotheses converge in a concrete example. Note how each hypothesis creates a Voronoi cell in the output space. In $\mathcal{L}_{WTA}$, a hypothesis is selected as the *winner* for a given $y \in \mathcal{A}(x)$, if y lies within the Voronoi cell defined by that hypothesis. This means that a hypothesis with an empty cell will always be a *loser* and, therefore, will not be moved. Consequently, as can be seen in Figure 4.2b, hypotheses that are initialized too far from any mode will not converge. Moreover, if a single hypothesis is consistently closest to 2 different modes, that hypothesis will be pulled in equal amounts towards both modes, making it converge to an averaged position. Both of these properties are problematic, since they can cause modes to remain uncovered, and cause stray hypotheses that become noise in the output.

RWTA

To combat these issues, Rupprecht et al. introduced a relaxed version of the WTA loss (RWTA). Rather than determining the loss strictly based on the closest hypothesis, a weighted sum is taken



(a) Initial hypotheses. (b) Converged hypotheses.

Figure 4.2: A visualization of the strict winner takes all training process. The domain represents the output space $\mathcal{Y}$. Given a single input x, a multiple-hypothesis prediction network outputs five hypotheses (blue markers). The input x can map to four ambiguous outputs $y \in \mathcal{A}(x)$ (red markers), which appear as separate modes in $p(y|x)$. Arrows indicate which hypothesis is the winner with respect to a $y \in \mathcal{A}(x)$. Lines portray the voronoi cells created by hypotheses, where dashed lines extend to infinity.

among all hypotheses as follows.

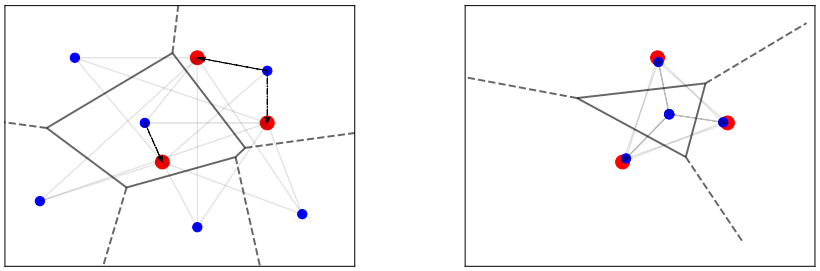
$$\mathcal{L}_{RWTA}(x, y) = \left(1 - \epsilon\frac{M}{M-1}\right) \underbrace{\min_{1 \leq j \leq M} \mathcal{L}\big(f_\theta^j(x), y\big)}_{\text{Winner term}} + \frac{\epsilon}{M-1} \underbrace{\sum_{j=1}^{M} \mathcal{L}\big(f_\theta^j(x), y\big)}_{\text{Sum term}} , \tag{4.16}$$

where ϵ, with $0 \leq \epsilon \leq 1$, is a weighting term. Note how a choice of $\epsilon = 0$ results in the strict WTA loss, and a choice of $\epsilon = 1$ removes the WTA concept completely. By adding the sum term, the RWTA loss function effectively forces every hypothesis to move at least slightly in the direction of all modes, even when its own Voronoi cell is empty. An illustrative example is given in Figure 4.3 for $|\mathcal{A}(x)| = 3$ and $M = 6$. The other issue with WTA, where a single hypothesis converges to an averaged position between two modes, is also largely solved. Eventually, one of the loser hypotheses is likely to pass close enough by one of the contending modes that it becomes a winner, thus freeing the other hypothesis to converge to the other mode.

The addition of the sum term does, however, come with its own unfortunate side-effects. Firstly, in the converged positions, each mode is still paired with only a single winner hypothesis. While the remaining losers are not stranded at their arbitrary initial positions, like in WTA, the sum term instead forces them all to a single shared equilibrium position (Figure 4.3b). Furthermore, since the winning hypothesis for one mode is also a loser for all other modes, the winning hypotheses converge slightly away from their paired mode. How far entirely depends on the choice of ϵ, with larger ϵ causing larger distances.

EWTA

Another variation of the WTA loss was introduced by Makansi et al. [40] called Evolving Winner Takes All (EWTA). The idea behind EWTA is to use an alternative relaxation of strict WTA, in which as many hypotheses as possible converge to one of the modes. In contrast to RWTA, where the eventual set of loser hypotheses converges to an averaged position. To this end, Makansi et al.



(a) Initial hypotheses. (b) Converged hypotheses.

Figure 4.3: A visualization of the relaxed winner takes all training process. The domain represents the output space $\mathcal{Y}$. Given a single input x, a multiple-hypothesis prediction network outputs six hypotheses (blue markers). The input x can map to three ambiguous outputs $y \in \mathcal{A}(x)$ (red markers), which appear as separate modes in $p(y|x)$. Arrows indicate which hypothesis is the winner with respect to a $y \in \mathcal{A}(x)$. Transparent arrows indicate pull resulting from the sum term. Lines portray the voronoi cells created by hypotheses, where dashed lines extend to infinity

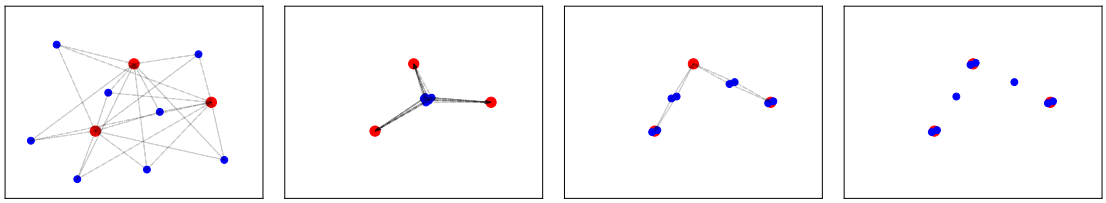
propose to let the k best hypotheses be winners, rather than only the single best. The parameter k is then 'evolved' from $k = M$ down to $k = 1$ over the course of the training process. Formally, we define the EWTA loss function as

$$\mathcal{L}_{EWTA}(x, y) = \min_{B \subseteq f_\theta(x), |B|=k} \sum_{a \in B} \mathcal{L}(a, y) \ , \tag{4.17}$$

where B is the set of k best hypotheses.

The exact inner workings of the EWTA training process are quite complicated to imagine, but we give an illustrative example in Figure 4.4. The process starts with k equal to the total number of hypotheses M. In the context of the figure, this implies $k = 8$. At this stage, the loss is computed between each hypothesis and each mode, causing all hypotheses to converge to a conditional average (Figure 4.4b). More intuitive is to imagine each mode *pulling* on all hypotheses. As k is decreased, each mode releases its pull on an increasing number of hypotheses, instead pulling only on the top-k closest hypotheses. In the example, for $k = 4$, the bottom left and bottom right modes each pull on 4 hypotheses (Figure 4.4c). Among these 4 hypotheses, 2 are free to converge to the respective mode, as they receive no pull from any other modes. The other 2, however, are also part of the 4 winning hypotheses for the top mode, thus making them converge to another average. Finally, the entire EWTA process converges when $k = 2$ (Figure 4.4d). In this configuration, each mode is able to pull on a unique set of 2 hypotheses, making those hypotheses converge entirely to the respective mode. Since the 3 modes are now only pulling on 6 distinct hypotheses in total, the 2 left over hypotheses receive no more pull at all. Consequently, they remain in their previous averaged positions.

In conclusion, EWTA provides an alternative way of relaxing the strict WTA loss. By avoiding the sum term in RWTA, no residual forces remain in the converged state. Consequently, loser hypotheses are not pulled to a shared averaged position, and winner hypotheses can fully converge to a ground truth mode. While stray hypotheses are not avoided entirely, their numbers are reduced,



(a) Initial hypotheses at $k = 8$. (b) Converged at $k = 8$. (c) Converged at $k = 4$. (d) Converged at $k = 2$.

Figure 4.4: A visualization of the evolving winner takes all training process. The domain represents the output space $\mathcal{Y}$. Given a single input x, a multiple-hypothesis prediction network outputs eight hypotheses (blue markers). The input x can map to three ambiguous outputs $y \in \mathcal{A}(x)$ (red markers), which appear as separate modes in $p(y|x)$. Arrows indicate which hypotheses belong to the k winners with respect to a $y \in \mathcal{A}(x)$. Overlapping markers are manually moved slightly apart to improve clarity. Figure based on [40].

making the distribution of hypotheses represent the ground-truth distribution more accurately.

4.2 Application to Orientation Estimation

As was mentioned at the start of this Chapter, the problem of ambiguity in orientation estimation relates very closely to multimodal output prediction. Concretely, we can define the input space $\mathcal{X}$ to be the space of all possible input images $\mathcal{I}$, and define the output space $\mathcal{Y}$ to be the rotation manifold $SO(3)$. Then, the mapping we wish to learn is one of images to rotations. Recall the definition of a *symmetry set* from Section 2.3.2

$$S(I) = \{R \in SO(3) : \mathcal{R}(R) = I\} \ . \tag{4.18}$$

The presence of global and occlusion-induced symmetries plays a dominating role in the characteristics of $S(I)$. Take, for example, an image I of an unobstructed and textureless cube. A cube has a discrete number of 24 rotational symmetries, such that $|S(I)| = 24$, where the notation $|\cdot|$ indicates the size of the set. An image of an unobstructed cone, on the other hand, would have an associated $S(I)$ with infinite elements, since a cone has a continuous axial symmetry. When considering occlusions, the situation gets even more complicated. Consider an image of a tea mug, which could either produce a finite or infinite set $S(I)$ depending on the view. If the ear of the mug is visible, the rotation is unambiguous and $|S(I)| = 1$. If the ear is occluded, the mug becomes symmetric around its central axis, causing $S(I)$ to become a set with infinite number of possible rotations.

As was explained earlier in Section 4.1, a network that produces only a single output will not be able to deal with this ambiguity. By applying multiple hypothesis prediction, however, we can create a model capable of finding multiple modes in the output distribution. This idea was first tested by Manhardt et al. [20], who applied RWTA training to a pose estimation network. In the rest of this Section, we explain how MHP models can be applied to orientation estimation. In the next chapter, this implementation is evaluated with various meta and embedded loss functions.

4.2.1 Embedded Loss Functions

Because the meta loss functions WTA, RWTA, and EWTA make no assumptions on the embedded loss function $\mathcal{L}$, applying them is as easy as choosing a suitable $\mathcal{L}$. In orientation estimation, such a loss function should capture a notion of distance between two rotation matrices. The minimal angular difference is a popular metric that can be used to this end [41]. It measures the smallest angle by which one rotation matrix can be rotated into another. Given two rotation matrices $R, R' \in SO(3)$, the minimal angular difference α is defined as

$$\alpha = \cos^{-1}\left(\frac{\mathrm{Tr}\left(RR'^{\intercal}\right) - 1}{2}\right) . \tag{4.19}$$

More formally, the minimal angular error describes the *geodesic distance* between R and R', as it gives the distance in radians between the two rotations on the rotation manifold. Using the minimal angular difference, we can formulate a loss function that directly satisfies our criterion. By renaming R to R_p, indicating a predicted rotation, and R' to R_{GT}, indicating the ground truth rotation, we formulate the *geodesic loss* as

$$\mathcal{L}_{GEO}(R_p, R_{GT}) = \cos^{-1}\left(\frac{\mathrm{Tr}\left(R_p R_{GT}^{\intercal}\right) - 1}{2}\right) . \tag{4.20}$$

The inverse cosine in this formulation is, however, inconvenient in practise. The expression inside can take values outside the range $[-1, 1]$ due to accumulating errors in floating point arithmetic. Instead, we can use the cyclic cosine loss [29] as a surrogate, which is defined as

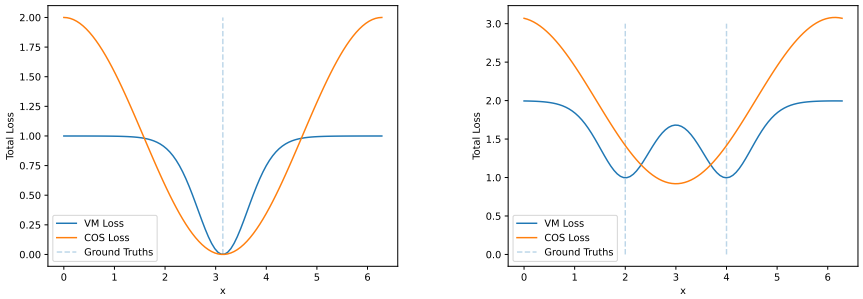
$$\begin{aligned} \mathcal{L}_{COS}(R_p, R_{GT}) &= 1 - \cos(\alpha) \\ &= 1 - \frac{\mathrm{Tr}\left(R_p R_{GT}^{\intercal}\right) - 1}{2} . \end{aligned} \tag{4.21}$$

Alternatively, Beyer et al. [29] proposed a loss function based on the Von Mises distribution. While the geodesic loss grows linearly with respect to the minimal angle between R_p and R_{GT}, this loss functions scales exponentially. Practically, it penalizes large errors more harshly than small errors. The loss is defined as

$$\begin{aligned} \mathcal{L}_{VM}(R_p, R_{GT}) &= 1 - e^{\mathcal{K}(\cos(\alpha)-1)} \\ &= 1 - e^{\mathcal{K}\left(\frac{\mathrm{Tr}\left(R_p R_{GT}^{\intercal}\right)-1}{2}-1\right)} , \end{aligned} \tag{4.22}$$

where $\mathcal{K}$ is a parameter analogous to the inverse of the standard deviation in a traditional Gaussian distribution. Much like $\mathcal{L}_{COS}$, $\mathcal{L}_{VM}$ does not require the computation of an inverse cosine.

What makes the Von Mises loss particularly interesting, is the shape of the loss function (Figure 4.5). The loss is minimized when the predicted rotation matches the ground truth, but converges to 1 for larger angular errors. Up until now, we have assumed that minimizing the loss for a single predicted output with respect to multiple ground truth outputs, results in that prediction converging to a single averaged position. This is also the case for $\mathcal{L}_{GEO}$, but, interestingly, not for $\mathcal{L}_{VM}$. Instead, if the ground truth rotations lie far enough apart, the Von Mises loss is minimized in multiple positions, each of which lies very close to one of the ground truths. A downside of using



(a) (b)

Figure 4.5: Plots illustrating the shape of both $\mathcal{L}_{GEO}$ and $\mathcal{L}_{VM}$. Functions are shown in a 2D orientation estimation setting for simplicity. Figure (a) plots both losses as a function of the predicted angle $x \in [0, 2\pi)$ (radians) with respect to a ground truth rotation of π radians. Figure (b) plots the sum of losses with respect to ground truth rotations 2 and 4 radians. $\mathcal{L}_{VM}$ uses $\mathcal{K} = 4$. For a detailed analysis see Appendix A.

the Von Mises loss is that its derivative converges to 0 for larger angular errors. Consequently, if a hypothesis is predicted with a large error, it produces very little gradient, damaging the training process. See Appendix A for more details.

4.2.2 Network Design

Based on previous works [34, 20], we implement a multiple hypothesis prediction model as a single neural network with duplicated output layers. The network can be seen as a predictor $f_\theta : \mathcal{I} \rightarrow SO(3) \times ... \times SO(3)$, that takes a single image $I \in \mathcal{I}$ as input and produces multiple rotations $f_\theta^1(I), ..., f_\theta^M(I) \in SO(3)$ as output.

Rather than designing the entire network from scratch, we rely on established high-performing network architectures to construct f_θ. We constrain images to be 224 pixels in both width and height and choose the popular Residual Network (ResNet) [42] architecture as the basis for this network. A number of ResNet versions are available that are 18, 34, 50, 101, and 152 layers deep. These models also come *pre-trained* on the image classification data set *ImageNet* [43], which consists of more than a million images with objects from 1000 categories. Re-using parameters from pre-trained networks in different contexts is called *transfer learning* and can kick-start a new training process.

Producing Orthogonal Output

Output rotations are represented as rotation matrices. That is, each hypothesis f_θ^i outputs a 3×3 matrix where the columns form an orthonormal basis. Outputting 3×3 matrices is easily achieved. Simply create an output layer with the same 3×3 shape, and duplicate it M times. Constraining these outputs to form orthogonal matrices, however, is not trivial. Without additional processing steps, the network would simply output the 9 values of a matrix independently, with no regard for the orthogonality constraint. To address this, a more elaborate scheme is used. Based on the work by Zhou et al. [41], a rotation is first output in the form of two $3D$ vectors x' and y'. Note

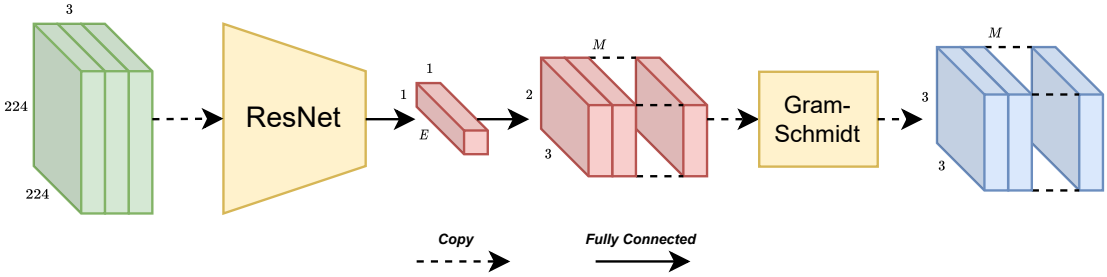


Figure 4.6: A schematic overview of the network architecture used in the multiple hypothesis prediction model for pose estimation.

that x' and y' are not normalized, and can be taken straight from the output of a neural network. A full set of local orthonormal axes is subsequently built by following the Gram-Schmidt process. More specifically, we first normalize $x = \frac{x'}{||x'||}$. Then, we compute $z' = x \times y'$ to find a vector that is perpendicular to the plane of x' and y'. Next, we also normalize $z = \frac{z'}{||z'||}$, and subsequently compute $y = z \times x$, which is orthogonal to both z and x. Note that y is automatically normalized, since z and x are orthogonal and normalized. Finally, this gives us three orthonormal axes that represent the predicted local axes of an object. These axes are converted to a rotation matrix R as

$$R = \begin{bmatrix} x_x & y_x & z_x \\ x_y & y_y & z_y \\ x_z & y_z & z_z \end{bmatrix} \in SO(3) \ .$$

Final Design

Putting everything together, the final architecture is depicted in Figure 4.6. In the first stage, an input image is fed to a ResNet model. The image consists of three channels (RGB) with 224×224 pixel values each. The output of the ResNet model is converted to an embedding of size E by a fully connected layer. This embedding is again fed to a fully connected layer, in order to produce M pairs of $3D$ vectors. Finally, by applying the Gram-Schmidt process, M proper rotation matrices are output.

Chapter 5

Experiments with Multi-Hypothesis Networks

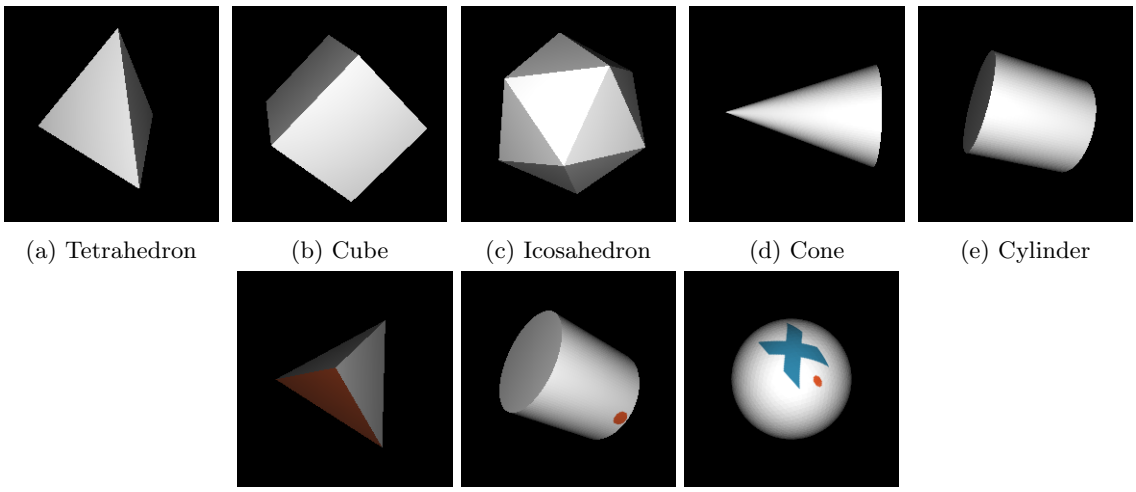
In this Chapter, we go over a set of experiments that aim to evaluate the various meta loss functions and embedded-loss functions described in the previous chapter. To this end, multiple models are implemented as described in Section 4.2.2, and trained using WTA, RWTA, and EWTA meta loss functions, together with cyclic cosine and Von Mises embedded loss functions. The performance of these models as multiple hypothesis orientation predictors is tested on a variety of objects. The choice of objects is based on the SYMSOL data set proposed in [18], and were specifically chosen for the types of symmetries they exhibit. In this evaluation, we are specifically interested in the ability of each model to predict hypotheses that fall within the symmetry set of an image.

First we explain in Section 5.1 how a data set can be created with annotated symmetry sets for every image, while covering different types of symmetries. Section 5.2 details the metrics used in evaluation. Next, Section 5.3 gives implementation details for the models that will be evaluated. Finally, results are presented and discussed in Section 5.4 .

5.1 Data Set

Evaluation should be performed on data that is relevant to the problems described in the thesis. As such, the images in these data sets should display *symmetry-induced ambiguity*. Moreover, ground truth annotations of symmetry sets should preferably be available for each image. This way, the output of all the considered models can be tested against a full set of symmetries. Furthermore, the models in this thesis only pertain to orientation estimation, rather than full 6D pose estimation. Preferably, the data sets reflect this, and do not consider translation. Finally, a data set preferably includes different types of symmetries. More specifically, we distinguish discrete and continuous symmetries for symmetric and nearly-symmetric objects, where nearly-symmetric objects can appear symmetric under self-occlusion.

This specific set of requirements greatly restricts which data sets can be used. The most restricting requirement is that of having annotated ground truth symmetry sets. Few data sets include this information. When they do, the annotations are limited to symmetric objects, and are



(a) Tetrahedron (b) Cube (c) Icosahedron (d) Cone (e) Cylinder

(f) Marked tetrahedron (g) Marked cylinder (h) Marked sphere

Figure 5.1: Example images from the SYMSOL data set. The top row shows the five symmetric objects in the SYMSOL I set. The bottom row shows the three nearly-symmetric objects from the SYMSOL II set.

not available for nearly-symmetric objects under occlusion. This is understandable, as capturing such information for real-life images is practically infeasible. Another restrictive requirement is that of only considering orientation. Many established pose estimation data sets exist [44, 10, 45, 46], but most cater towards full 6D pose estimation. Translation could be taken out of the equation by cropping images based on their bounding box within each image, but concerns are that this may cause distortion which influences the perceived orientation of the object. If the object is not centered, it is observed from a slight angle. Nevertheless, some related work has used this strategy [17].

5.1.1 SYMSOL

Considering all requirements, only the SYMmetric SOLids (SYMSOL) data set by Murphy et al. [18] comes close. It contains two subsets: SYMSOL I and SYMSOL II, containing symmetric and nearly-symmetric objects respectively. The objects are synthetically rendered and are placed in the center of each image, thus ignoring translation. Objects are observed from a stationary camera and under constant lighting (Figure 5.1). More details on the two subsets are as follows.

1. **SYMSOL I**
   Contains five objects with global symmetries: *tetrahedron*, *cube*, *icosahedron*, *cone*, and *cylinder*. The tetrahedron, cube, and icosahedron each have 12, 24, and 60 discrete symmetric rotations respectively, whereas the cone and cylinder have continuous symmetries. The cone is symmetric around a single axis, while the cylinder can also be rotated 180 degrees. A total of 50000 images are available for each object, where 45000 are used for training and 5000 are used as a test set. The objects appear in orientations randomly sampled from $SO(3)$. SYMSOL I also comes with annotated symmetry sets for every image, as they are easily derived for these shapes. For the cone and cylinder, the continuous ranges of symmetric

rotations are discretized at 1 degree.

2. **SYMSOL II**
   Contains nearly-symmetric objects that exhibit symmetries under self-occlusion. These include a *marked tetrahedron* (tetX), *marked cylinder* (cylO), and *marked sphere* (sphereX). As the names suggest, each of these images has symmetry-breaking markings applied to its surface. The tetrahedron has a single colored face, the cylinder is marked with a dot, and the sphere is marked with both an X and an adjacent dot. Depending on whether the markings are visible, the symmetry set of an image can be very different.

Tetrahedron For the tetrahedron, if the colored face is visible, only 3 discrete symmetries remain. If the colored face is not visible, then the symmetry set will contain 3, 6, or 9 discrete symmetries if 3, 2, or 1 non-colored face(s) is/are visible respectively.

Cylinder For the cylinder, if the mark is visible, then there is no ambiguity left at all. If the mark is not visible, then only the subset of symmetric rotations for a normal cylinder remains in which the mark does not enter the view of the camera.

Sphere For the sphere, the situation is more complicated. If the X and dot are both visible, then there is no ambiguity. If neither is visible, then only the symmetries of the sphere remain in which neither of the markings would be visible. If only one of the markings is visible, a complicated set of discrete or continuous symmetries appears, that can not be easily described nor imagined.

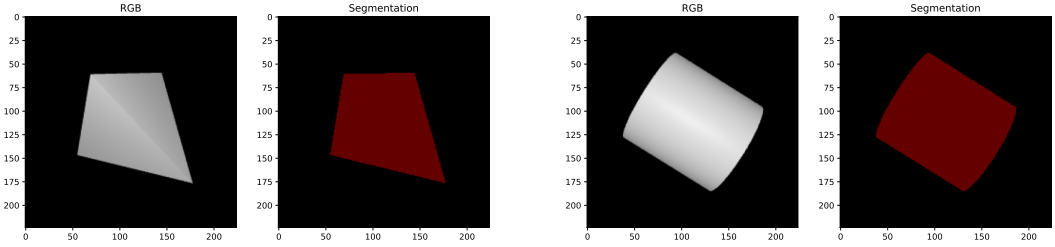
As with SYMSOL I, a total of 50000 images are available for each object, where 45000 are used for training and 5000 are used as a test set. However, since symmetry sets for these nearly-symmetric objects are complex and view-dependent, annotations do not include full symmetry sets.

Notably, both SYMSOL I and II only contain synthetically rendered images. Moreover, images show objects in a simplistic setting, using constant lighting conditions and perfectly centered objects. Compared to real life images, or more realistic synthetic images [6], this simplistic setting simplifies the orientation estimation problem in the sense that a neural network is expected to learn effective feature-extraction more easily, and with fewer network parameters. Nevertheless, specifically in SYMSOL II, there exists a complex relationship between viewpoint and symmetry set. Since we are most interested in the ability of MHP models to learn this relationship, rather than their ability to perform complex feature extraction, we still consider SYMSOL to be a suitable data set for this evaluation.

Though SYMSOL II does not contain annotated symmetry sets like SYMSOL I, it would be a valuable addition. Having the symmetry sets available would create the first data set covering symmetry under self-occlusion with full symmetry annotations. Such a data set is also a necessity for evaluation any model's performance on these types of objects. Hence, we make a custom version of SYMSOL II with symmetry set annotations.

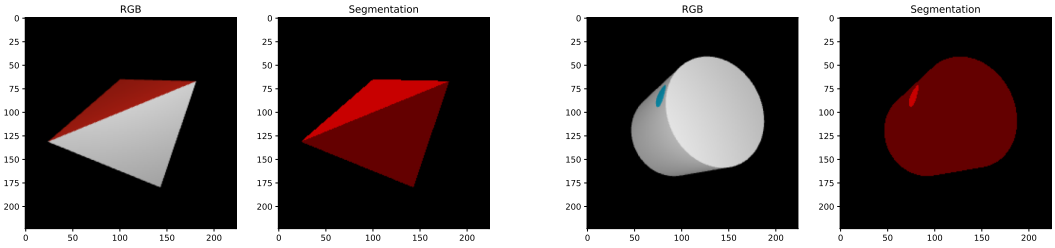
5.1.2 Annotating SYMSOL II

As was explained in the previous section, the nearly-symmetric objects in SYMSOL II have symmetries that differ drastically between views. More specifically, the visibility of the markings



(a) TetX with no visible marking.

(b) CylO with no visible marking.



(c) TetX with visible marking.

(d) CylO with visible marking.

Figure 5.2: Example images from the annotated recreation of SYMSOL II. Both RGB images as well as the associated segmentation map are shown. In the segmentation maps, the segmentation index of the marking is mapped to a light red color, while the segmentation index of the body is mapped to a dark red color.

determines the shape of symmetry sets in a complex way. Nevertheless, some structure can be found. For a given image, the symmetry set for a marked object is a subset of the symmetry set of the equivalent unmarked object in the same orientation. Since symmetry sets for the unmarked objects are trivial to compute, they can be used as a set of candidate rotations for the marked object. If a procedure can be devised to test whether a candidate from the unmarked object's symmetry set is also in the marked object's symmetry set, then the symmetry set of the marked object can be computed.

One complication in this regard is that continuous symmetries would produce an infinite number of candidate rotations. However, this is already addressed in SYMSOL I, where continuous rotations are discretized at 1 degree. For the cylinder, this reduces the number of candidate rotations to 720, which is a manageable amount. Unfortunately, this discretization does not do much for the marked sphere. An unmarked sphere is completely symmetric under all rotations, meaning that it produces symmetry sets covering all of $SO(3)$. Discretizing $SO(3)$ at a resolution of 1 degree would create millions of candidate rotations. Therefore, we only consider the marked tetrahedron and marked cylinder, and drop the marked sphere from the data set.

Rendering Images and Segmentation Maps

To compute symmetry set annotations, we exploit the rendering of *segmentation maps*. When rendering an image, the geometry of objects in a virtual world is rasterized to that image, meaning that for each pixel it is computed which piece of geometry covers that pixel in the final image. Hence, for each pixel it is known which, if any, piece of geometry covers that pixel. A segmentation map returns exactly this information. Firstly, each piece of geometry is assigned an integer *segmentation index*. Then, a segmentation map can be rendered by assigning to each pixel the

segmentation index of the geometry covering that pixel in the image (Figure 5.2).

Now, to recreate an annotated version of SYMSOL II, we use the following procedure. First, the marked tetrahedron and marked cylinder objects were recreated in Blender[1]. In these models, the geometry constituting the marking is separated from the rest of the body in each object. The marking and the rest of the geometry are given a different segmentation index, such that for any pixel in a rendered segmentation map, it can be established whether that pixel belongs to the marking or the object's body.

We now render a data set of 50000 RGB images for both the marked cylinder and marked tetrahedron, each split into 5000 test images and 45000 training images. To this end, the objects are placed at the origin of a virtual world, with a camera observing from the negative z-axis. A light is placed at the same position as the camera, such that the observed part of the object is always lit. When rendering an image I, a rotation R is sampled randomly from $SO(3)$ using the method described in [47], and is applied to the object. The object is then rendered from the point of view of the camera using OpenGL[2] to an image 224 by 224 pixels in size. The pair (I, R) now contains an image I along with the associated ground truth orientation R of the object in that image.

Computing Symmetry Sets

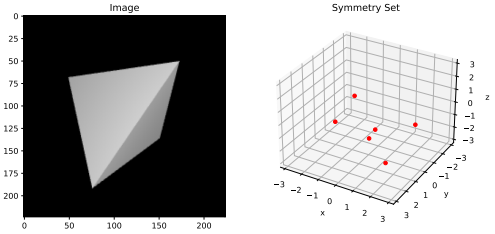
Finally, the most interesting part is computing the symmetry set annotations. This process is implemented slightly differently for the cylinder and the tetrahedron. Moreover, because this method can be computationally intense, we only provide symmetry set annotations for test images. Examples of the results are shown in Figure 5.3.

Marked Cylinder Firstly, for the cylinder, given an image-orientation pair (I, R_{GT}), we render a segmentation map I_s of the same object in the same orientation R_{GT}. This is, again, done using OpenGL. Then, we check whether the marking is visible in I_s by testing each pixel value against the segmentation index of the marking. If the marking is visible, then there is no ambiguity in the image, and the symmetry set $S(I) = \{R\}$. If, on the other hand, the marking is not visible, then there is ambiguity, and we proceed to the next step. Let $C(R_I)$ be the set of discretized symmetric rotations of a cylinder in the identity rotation R_I. Note that a cylinder has a full 360 degree rotational symmetry about its central axis, and can additionally be flipped end-to-end. Hence, discretized at 1 degree, $|C(R_I)| = 720$. Now, we define the same set of symmetric rotations for a cylinder in rotation R_{GT} as

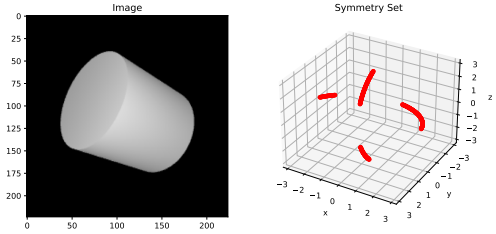
$$C(R_{GT}) = \{R_{GT}R : R \in C(R_I)\} \ . \tag{5.1}$$

Hence, $C(R_{GT})$ forms the set of candidate rotations for the symmetry set $S(I)$. For each candidate rotation $R_c \in C(R_{GT})$, a new segmentation map I_s^c is rendered with the object in rotation R_c. The symmetry set $S(I)$ now consists of those rotations R_c, for which the marking is not visible in I_s^c. Let i be the segmentation index of the marking, and consider I_s^c as a set of pixel values, then

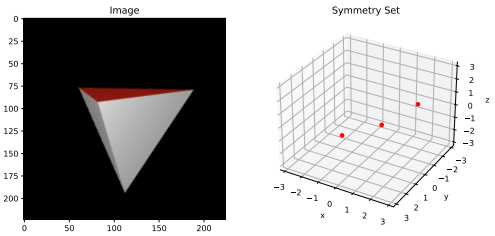
[1] https://www.blender.org/
[2] https://www.opengl.org/



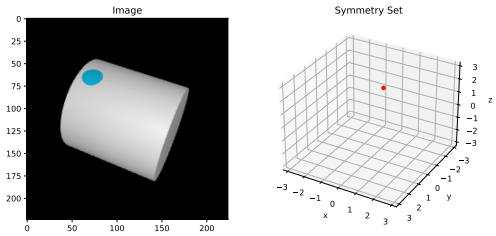
(a) TetX with two gray faces visible.



(b) CylO with no visible marking.



(c) TetX with visible marking.



(d) CylO with visible marking.

Figure 5.3: Example symmetry sets from the annotated recreation of SYMSOL II. Images are shown with their respective symmetry set represented in rotation vectors, which ate illustrated in a π-ball plot (Section 2.1.3).

we formally define

$$S(I) = \{R_c \in C(R_{GT}) : i \notin I_s^c\} \ . \tag{5.2}$$

Marked Tetrahedron In contrast to the marked cylinder, images of the marked tetrahedron are always ambiguous, with at least three symmetric rotations. Again, let (I, R_{GT}) be an image-orientation pair, and I_s be a segmentation map of the object in rotation R_{GT}. Let $C(R_I)$ be the set of 12 symmetric rotations for a tetrahedron in the identity rotation, and define $C(R_{GT})$ as in Equation 5.1. We again distinguish two cases. Firstly, if the colored face is visible in I_s, then $S(I)$ contains only the three rotations for which the colored face rotates onto itself. These three rotations are manually identified as the subset $C'(R_I) \subset C(R_I)$, before the data generation process. Hence, the symmetry set for I can be trivially computed as

$$S(I) = \{R_{GT}R : R \in C'(R_I)\} \ . \tag{5.3}$$

If the colored face is not visible in I_s, then the symmetry set is computed similar to the marked cylinder. A segmentation map I_s^c is rendered for each candidate rotation $R_c \in C(R_{GT})$. The symmetry set now consists of those rotations for which the marking does not become visible:

$$S(I) = \{R_c \in C(R_{GT}) : i \notin I_s^c\} \ . \tag{5.4}$$

In practise, this latter step is susceptible to slight numerical errors in floating-point computation. As a consequence, a proper symmetric rotation $R \in S(I)$ may render to a segmentation map in which a handful of pixels of the colored face become visible. To deal with this, a tolerance is added. More specifically, let p be the number of pixels in I_s^c that belong to the marking, then R_c is accepted if $p < \mathcal{T}$, where $\mathcal{T} = 5$ was empirically determined to be a good threshold.

5.2 Evaluation Metrics

The set of output hypothesis of an MHP model should be accurate and diverse. This Section will describe quantitative metrics by which MHP models can be evaluated to test them on these properties. These metrics fall into two categories, measuring *recall* and *precision*.

5.2.1 Recall

Given an image I and an associated symmetry set $S(I)$, recall describes how much of $S(I)$ is covered by the hypotheses of an MHP model. Depending on the shape of $S(I)$, perfect recall can be defined in different ways. Let M be the number of hypotheses, then if $|S(I)| \leq M$, it is possible for the hypotheses to cover the entire symmetry set. This happens when an observed object has a discrete number of symmetric rotations, in which case it is desirable to have all these symmetries covered.

If, on the other hand, $|S(I)| > M$, then it is not possible to cover all elements of the symmetry set. This happens in any case where the observed object as has a continuous symmetry, such that $|S(I)| = \infty$. In this case, it is most desirable that the hypotheses spread out evenly across the potentially multiple ranges of $SO(3)$ that are spanned by the symmetry set.

A quantitative measure of both the aforementioned definitions is best given by the distance between members of the symmetry set, and the closest hypothesis. Ideally, the distance from each rotation in the symmetry set to the closest respective output hypothesis is minimal. More formally, let $f_\theta^i(I)$ be the i^{th} output hypothesis, then the quantity describing how well any $R \in S(I)$ is covered, is given by

$$\min_{1 \leq i \leq M} d\left(R, f_\theta^i(I)\right) \ , \tag{5.5}$$

where the distance metric $d(\cdot, \cdot)$ is defined as the minimal angular difference, or angular error, between two rotation matrices

$$d(R_x, R_y) = 2 \cdot \sin^{-1}\left(\frac{|R_y - R_x|_F}{2\sqrt{2}}\right) \ . \tag{5.6}$$

Note how this definition of the minimal angular difference is different from what was introduced back in Equation 4.19. Brégier [28] found that implementing the formula in Equation 4.19 produces sizable numerical errors when the minimal angular error is small. The *Rotation Manipulation* (*RoMa*)[3] library introduced in the paper, implements the formula from Equation 5.6 instead. Moreover, it is shown to have superior precision. Hence, we adopt it in this evaluation, to get more precise results.

Using the measure of recall for a single rotation mentioned above, a metric can now be defined to measure recall on an entire data set. Let $\mathcal{D}_{test}$ be a data set containing pairs $(I, S(I))$ of test images and their respective discretized symmetry set. The recall of an MHP model on $\mathcal{D}_{test}$ is given by the mean angular error between members of the annotated symmetry sets and their closest respective output hypothesis. More formally, let L be the sum of the number of symmetries

[3] https://naver.github.io/roma/

$|S(I)|$ over all images $I \in \mathcal{D}_{test}$, then the recall score on $\mathcal{D}_{test}$ is computed as

$$\frac{1}{L} \sum_{(I,S(I))\in\mathcal{D}_{test}} \sum_{R\in S(I)} \min_{1\leq i\leq M} d\left(R, f_\theta^i(I)\right) \ . \tag{5.7}$$

Alternatively, the recall can be quantified as the ratio of symmetry set members that are considered 'covered'. Here, a symmetry set member is considered 'covered' if an output hypothesis lies within a certain maximum distance $\mathcal{T}$. We will refer to this ratio as the *coverage at* $\mathcal{T}$. However, note that he coverage is only useful for objects with discrete symmetries. In the case of a continuous symmetry, the hypotheses can impossibly cover the entire symmetry set. Therefore, the coverage may be low, even when the hypotheses are perfectly spread out.

5.2.2 Precision

Precision in this context relates to the number of stray hypotheses generated by a model. Ideally, every hypothesis is meaningful, meaning that it is close to some member of the symmetry set. It can, therefore, be quantified similarly to recall. Again, let $f_\theta^i(I)$ be the i^{th} output hypothesis for an image I, then the precision of $f_\theta^i(I)$ is given by the quantity

$$\min_{R\in S(I)} d\left(R, f_\theta^i(I)\right) \ , \tag{5.8}$$

where the distance metric $d(\cdot,\cdot)$ is again defined as the minimal angular difference.

Precision on an entire data set is again defined as the mean angular error between all hypotheses and the closest member of the respective ground truth symmetry set. Let D_{test} be the data set, this is formally defined as

$$\frac{1}{|\mathcal{D}_{test}|} \sum_{(I,S(I))\in\mathcal{D}_{test}} \frac{1}{M} \sum_{i=1}^{M} \min_{R\in S(I)} d\left(R, f_\theta^i(I)\right) \ . \tag{5.9}$$

Moreover, the precision can again be expressed as a ratio of 'correct' hypotheses, where a hypothesis is accepted as correct if it lies within a maximum distance of $\mathcal{T}$ from the ground truth symmetry set. We will refer to this ratio as the *accuracy at* $\mathcal{T}$. Unlike the earlier defined coverage, the accuracy can be used for objects with either discrete or continuous symmetries.

5.3 Implementation Details

The MHP models were implemented in PyTorch[4], according to the network design detailed in Section 4.2.2. The 18-layer version of ResNet was used as the first stage of the network, which is the smallest available size. Given that the images in the SYMSOL dataset show objects in a simplistic synthetic setting, this size is sufficient to achieve good performance. The red, green, and blue channels of images are normalized using mean 0.485, 0.456, and 0.406 and standard deviation 0.229, 0.224, and 0.225 respectively, before being input to the ResNet model, with the purpose of being compatible with pre-trained versions of ResNet available in PyTorch[5]. Finally, the output of

[4] https://pytorch.org/

[5] `https://pytorch.org/hub/pytorch_vision_resnet/`

the ResNet model is converted to the embedding by a fully connected layer with ReLU activation, where the embedding size E was fixed at 256.

5.3.1 Hyperparameters

The number of hypotheses M is chosen based on the objects in the data set. The most important criteria is that, for the objects with discrete symmetries, M should be larger than the number of symmetric rotations, such that each symmetry can be covered. Moreover, since the number of generated stray hypotheses is of particular interest when comparing the various meta loss functions, there should be more hypotheses than symmetries for the discrete objects. Furthermore, the number of hypotheses is somewhat limited by the EWTA training process. The parameter k has to be decreased during training. The more hypotheses, the more steps have to be taken in decreasing k. Considering these constraints, it was decided to fix M at 30 for all objects but the icosahedron, for which M is set to 75. The choice of $M = 30$ is enough to cover all 24 symmetries of the cube with 6 hypotheses to spare, and can cover the 12 symmetries of the tetrahedron more than twice. Setting $M = 75$ for the icosahedron covers all 60 symmetries with 15 to spare.

Separate networks are trained using each combination of WTA, RWTA, or EWTA meta loss with cyclic cosine of Von Mises embedded loss, for each object in the SYMSOL data set. For RWTA, ϵ was set to 0.05, and for the Von Mises loss, $\mathcal{K}$ was set to 4. For EWTA, the parameter k is initialized to M, and then decreased using the process described in the supplemental materials of [40]. More specifically, k is halved every few epochs during training, until it reaches $k = 1$. Several extra epochs are spent while $k = 1$, to give the network a chance to converge the hypotheses to their final position. Notably, it is not quite clear in the literature how the process of decreasing k is best implemented. One alternative is to decrease k linearly, as implemented in [13], but the quality of results was found to vary greatly.

5.3.2 Training

Finally, the process of training the networks differs slightly between the SYMSOL I and SYMSOL II data sets.

1. **SYMSOL I**
   For WTA and RWTA, networks were trained for 30 epochs using a batch size of 64. Training for additional epochs was not found to further improve results. Network parameters are optimized using the ADAM [48] optimizer. The learning rate starts at $1e^{-3}$, and is decreased to $1e^{-4}$ and $1e^{-5}$ after epochs 15 and 25 respectively. Decreasing the learning rate in this way allows for small adjustments to the network parameters, improving convergence. For EWTA, an extra 5 epochs of training were given, such that networks are trained for a total of 35 epochs. The parameter k is halved every few epochs, and is set to hit $k = 1$ after epoch 20. The learning rate starts at $1e^{-3}$ and is decreased to $1e^{-4}$ after epoch 30.

2. **SYMSOL II**
   The objects in SYMSOL II exhibit more complicated symmetries. Consequently, networks trained on SYMSOL II were found to require more training time than their SYMSOL I counter parts. For WTA and RWTA, networks were trained for 50 epochs with a batch size

of 64. Network parameters are optimized using the ADAM optimizer. The learning rate starts at $1e^{-3}$, and is decreased to $1e^{-4}$ and $1e^{-5}$ after epochs 25 and 40 respectively. For EWTA, no extra epochs were given. The parameter k is, again, halved every few epochs, and is scheduled to hit $k = 1$ after epoch 30. The learning rate starts at $1e^{-3}$ and is decreased to $1e^{-4}$ after epoch 42.

5.4 Evaluation

As was explained in Section 4.1.2, the standard WTA training scheme has two main problems: (1) a single hypothesis can consistently lie closest to multiple different symmetries, making that hypothesis converge to an averaged position between these symmetries, and (2) hypotheses that do not lie closest to any symmetry are never moved. The first problem concerns recall, as it can leave several symmetries uncovered. The second problem concerns precision, since the stranded hypotheses can be inaccurate. Results are shown for metrics regarding recall and precision in Table 5.1 and 5.2 respectively.

			SYMSOL I					SYMSOL II	
			tet	cube	icosa	cone	cyl	tetX	cylO
mean error (↓)	WTA	$\mathcal{L}_{COS}$	0.018	0.038	0.038	0.070	0.109	0.010	0.088
		$\mathcal{L}_{VM}$	0.016	0.035	0.030	0.075	0.111	0.010	0.084
	RWTA	$\mathcal{L}_{COS}$	0.018	0.041	0.041	0.067	0.112	0.020	0.236
		$\mathcal{L}_{VM}$	0.017	0.032	0.029	0.073	0.109	0.012	0.171
	EWTA	$\mathcal{L}_{COS}$	0.044	0.066	0.101	0.066	0.116	0.018	0.127
		$\mathcal{L}_{VM}$	0.032	0.038	0.077	0.063	0.114	0.022	0.154
coverage at 5 (↑)	WTA	$\mathcal{L}_{COS}$	0.997	0.948	0.957	-	-	1.000	-
		$\mathcal{L}_{VM}$	0.997	0.956	0.976	-	-	0.999	-
	RWTA	$\mathcal{L}_{COS}$	0.997	0.956	0.976	-	-	0.995	-
		$\mathcal{L}_{VM}$	0.995	0.968	0.977	-	-	0.999	-
	EWTA	$\mathcal{L}_{COS}$	0.942	0.818	0.593	-	-	0.998	-
		$\mathcal{L}_{VM}$	0.977	0.959	0.770	-	-	0.991	-

Table 5.1: Recall metrics evaluated on the SYMSOL I and SYMSOL II data sets. The best and second best performing networks for each object have a dark gray and light gray colored cell respectively. Mean errors are given in radians, lower is better (↓). Coverage is calculated at 5 degrees, higher is better (↑).

5.4.1 SYMSOL I

Continuous Symmetries

Scores on the cone and cylinder objects, which strictly exhibit continuous symmetries, are close between all methods. This is true for both metrics concerning recall, as well as precision. Symmetry sets for images of these objects are infinite, consisting of continuous ranges of rotations. Considering that there is only a finite number of 30 hypotheses, there are not nearly enough hypotheses to cover all symmetries. In this case, we expect the hypotheses to spread out equally over the ranges of symmetries. The better the the spread of hypotheses, the lower the expected

			SYMSOL I					SYMSOL II	
			tet	cube	icosa	cone	cyl	tetX	cylO
mean error (↓)	WTA	$\mathcal{L}_{COS}$	0.228	0.079	0.081	0.026	0.015	0.917	0.948
		$\mathcal{L}_{VM}$	0.235	0.070	0.083	0.079	0.017	0.910	0.988
	RWTA	$\mathcal{L}_{COS}$	0.240	0.086	0.117	0.016	0.023	0.540	0.069
		$\mathcal{L}_{VM}$	0.168	0.072	0.080	0.037	0.016	0.224	0.684
	EWTA	$\mathcal{L}_{COS}$	0.169	0.132	0.157	0.019	0.029	0.255	0.193
		$\mathcal{L}_{VM}$	0.063	0.056	0.135	0.016	0.024	0.210	0.090
accuracy at 5 (↑)	WTA	$\mathcal{L}_{COS}$	0.550	0.839	0.828	0.984	0.996	0.299	0.534
		$\mathcal{L}_{VM}$	0.548	0.852	0.836	0.940	0.995	0.296	0.538
	RWTA	$\mathcal{L}_{COS}$	0.536	0.822	0.781	0.997	0.985	0.289	0.816
		$\mathcal{L}_{VM}$	0.647	0.854	0.841	0.959	0.996	0.560	0.665
	EWTA	$\mathcal{L}_{COS}$	0.656	0.661	0.479	0.996	0.980	0.565	0.631
		$\mathcal{L}_{VM}$	0.870	0.898	0.628	0.998	0.989	0.648	0.762

Table 5.2: Precision metrics evaluated on the SYMSOL I and SYMSOL II data sets. The best and second best performing networks for each object have a dark gray and light gray colored cell respectively. Mean errors are given in radians, lower is better (↓). Accuracy is calculated at 5 degrees, higher is better (↑).

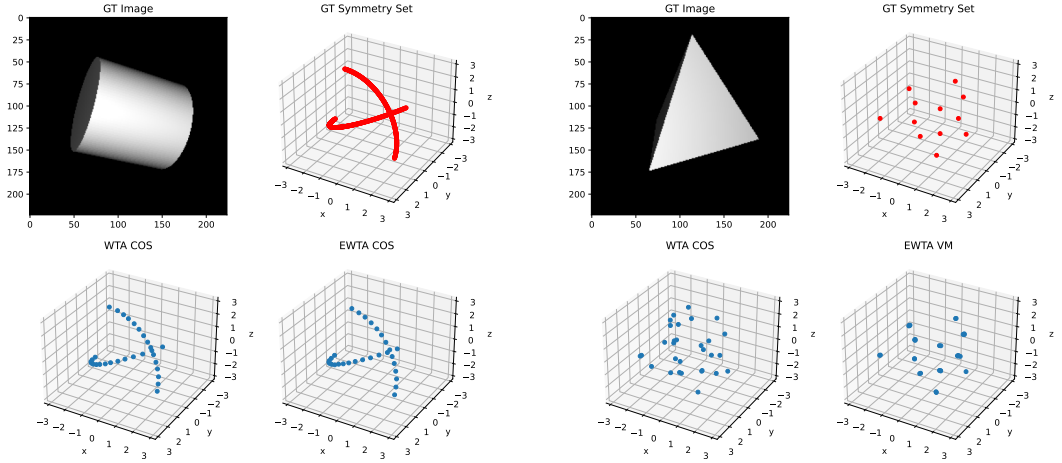
angular error between any ground truth symmetry and the nearest hypothesis. Therefore, the quality of the spread is best measured by the mean error metric describing recall.

Results show that this desired effect is achieved by all methods (Figure 5.4a). Mean errors for recall are consistently low for all training strategies. Differences in scores are slightly more pronounced on the cone object, with EWTA in combination with the Von Mises loss slightly outperforming the rest. The methods that perform worse, like WTA and RWTA in combination with the Von Mises loss, also show slightly worse accuracy at 5 degrees. This indicates the presence of stray hypotheses, meaning that fewer hypotheses are left to cover the range of symmetries, in turn explaining the higher mean errors for recall.

Discrete Symmetries

The differences between methods are much more pronounced on the tetrahedron, cube, and icosahedron objects, which exhibit strictly discrete symmetries.

Recall The difference between using the cyclic cosine loss and Von Mises loss is quite small, with the Von Mises loss performing marginally better. Differences between the training schemes, on the other hand, are slightly more pronounced. While WTA and RWTA show very comparable results, EWTA performs notably worse, with the icosahedron being the most significant. This is likely due to the inconvenient process of lowering the hyperparameter k during training. Only once k reaches 1 can the hypotheses start to converge to their final positions. Since this only happens after epoch 20 out of the 35 training epochs, there is comparably less time to converge when compared to WTA and RWTA. Increasing the time spent at $k = 1$ is, however, not a general solution to this problem. Doing so emulates the standard WTA training process, where loser hypotheses no longer contribute to the loss. In turn, this can harm the precision of the model. The scores, furthermore, show that this problem more pronounced when using cyclic cosine loss



(a) (b)

Figure 5.4: Examples of hypotheses predicted on the SYMSOL I data set. Both the best performing (WTA COS) and worst performing (EWTA COS) models are able to correctly find the two continuous ranges of symmetries for the cylinder (a). On objects with discrete symmetries, precision varies drastically between training strategies (b).

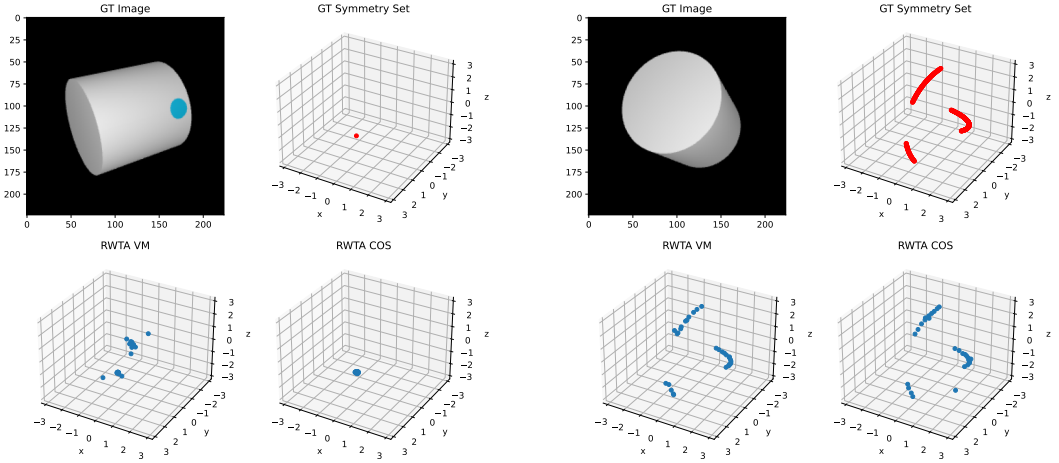
compared to the Von Mises loss.

Overall, the best performance with respect to recall seems to be achieved using either WTA or RWTA together with the Von Mises loss function. The comparable scores between WTA and RWTA are interesting. Firstly, this suggests that the problem of hypotheses converging to averaged positions in between symmetries rarely appears. Secondly, the influence of the ϵ term in RWTA, pulling even the winning hypotheses towards a global averaged position, seems to be small.

Precision Scores for metrics concerning precision show much more significant differences between methods. EWTA in combination with the Von Mises loss shows by far the best performance on the tetrahedron and cube, approaching almost 90% accuracy at 5 degrees (Figure 5.4b). Performance is, however, much worse on the icosahedron, presumably due to the same problem mentioned above.

Furthermore, for RWTA and EWTA, the Von Mises loss consistently outperforms the cyclic cosine loss by a large margin. This can be explained by the properties of the Von Mises loss mentioned in Section 4.2.1 and Appendix A. More specifically, the Von Mises loss does not force hypotheses to an averaged position, when summing the loss of with respect to multiple symmetries. Using the cyclic cosine loss, the surplus hypotheses converge to an average position, whereas, using the Von Mises loss, they converge to positions close to the symmetries. As a consequence, the precision significantly increases. A similar effect occurs when using EWTA, where high values of k would normally cause hypotheses to converge to averaged positions.

Finally, it is interesting to note how RWTA does not significantly increase precision compared to standard WTA. As explained in Section 4.1.2, the relaxation in RWTA causes all surplus hypotheses to converge to a global averaged position. Since this position is not guaranteed to lie close to any symmetry, the accuracy of the hypotheses is not expected to increase. In fact, it may even harm the precision, as can be seen in the mean error and accuracy at 5 degrees for the RWTA



(a) (b)

Figure 5.5: Plots comparing two models trained using the Von Mises loss and cyclic cosine loss in combination with RWTA. On the marked cylinder object, the Von Mises loss fails to cluster hypotheses on images without ambiguity, in contrast to the cyclic cosine loss (a). Nevertheless, both methods are able to cover broken symmetries (b).

model using cyclic cosine loss. EWTA, on the other hand, was specifically introduced to increase precision, which shows in the results.

5.4.2 SYMSOL II

Marked Tetrahedron

Compared to the standard tetrahedron from SYMSOL I, it is more difficult to achieve high precision on the marked tetrahedron. This is because, depending on the view, only a subset of the 12 tetrahedral symmetries remain. A model that outputs all 12 symmetries for every image, while having a high recall, will therefore have poor precision. This is partly what happens for the WTA models, which achieve the highest recall, but perform by far the worst on precision.

RWTA and EWTA also achieve high recall, but, moreover, greatly improve on precision. Again, EWTA in combination with the Von Mises loss outperforms all other methods. Furthermore, consistent with results on SYMSOL I, the Von Mises loss function performs significantly better when using either RWTA or EWTA. The difference is particularly apparent for RWTA.

One observation of particular interest concerns the precision of the RWTA model using cyclic cosine loss. While this model significantly improves the mean error with respect tot WTA, the accuracy at 5 degrees actually decreases. This is again an example of the surplus hypotheses reaching an averaged position, which may lie relatively close to the ground truth symmetries, but not close enough such that the hypotheses could be considered accurate.

Marked Cylinder

Finally, the marked cylinder is a particularly interesting case. Depending on the view, there is either no symmetry at all, or a continuous symmetry like that of a cylinder. An ideal model,

therefore, clusters all hypotheses together in the former case, and spreads the hypotheses evenly across the continuous ranges of symmetries in the latter case.

Differences between the various methods are again stark. Firstly, the WTA models perform significantly better on recall, but have very poor precision. This poor precision is no surprise, considering the properties of the WTA training process. For images in which the marking is visible, and no symmetry exists, only a single hypothesis converges to the ground truth rotation. The rest land in arbitrary positions in the output space. RWTA and EWTA models score slightly worse on recall, but have significantly better precision. Interestingly, for RWTA, the cyclic cosine loss achieves significantly better precision than the Von Mises loss. Upon further inspection, the models using the Von Mises loss were found to poorly cluster hypotheses for images without ambiguity. This is explained by the vanishing of gradients for hypotheses with large errors, as these get stuck in positions far away from the single ground truth rotation (Figure 5.5a).

5.4.3 Summary

In general, the use of RWTA does not give much benefit over using the standard WTA strategy. The problem that the relaxation term solves, which is that of hypotheses converging to averaged positions, does not seem to have a large impact in pose estimation, considering the results shown above. Consequently, the recall is not generally improved by using RWTA. Moreover, the effect of having surplus hypotheses cluster in a globally averaged position, which is caused by the added relaxation, does not increase precision. On the contrary, the accuracy of hypotheses can be worse.

The EWTA training scheme, on the other hand, shows a lot of potential. Results show how the precision of hypotheses is greatly improved on objects with discrete symmetries. Nevertheless, the training process is difficult to control. The hyperparameter k has to be decreased over the course of the training process, yet it is unclear how this is best done. Every time k changes value, the hypotheses may converge to completely different positions. Hence, more training time is needed. Furthermore, the optimal time spent on each value of k differs between objects. Manual tuning should be applied to achieve optimal results for each individual object. In a setting where no knowledge about possible symmetries is assumed, this is an undesirable property. A user in this position should run multiple experiments with strategies for setting k and decreasing it. Alternatively, RWTA could be used instead, as it does not require the same parameter tuning.

Finally, the Von Mises loss significantly increases precision when dealing with discrete symmetries. This difference is due to its behaviour when a hypothesis receives loss from multiple symmetries in RWTA and EWTA. In contrast to the cyclic cosine loss, which makes such hypotheses converge to an averaged positions, the Von Mises makes them converge to one of the symmetries. In turn, this greatly improves precision. The only exception is seen among the RWTA models trained on the marked cylinder. Here, vanishing gradients in the Von Mises loss cause poor clustering of hypotheses on images without ambiguity.

Chapter 6

A Continuous Representation

In the following chapter, we propose a new network design in which we model orientation estimation as a binary classification problem. Context and motivation are given in Section 6.1. Next, the concept of an occupancy network is detailed in Section 6.2, which forms the basis of our proposed design. Finally, in Section 6.3 we apply occupancy networks to orientation estimation.

6.1 Overview

Though an MHP model is able to capture multiple modes in the output distribution, it produces only a discrete number of outputs. However, when an object exhibits a continuous symmetry in an image I, we know that there are infinitely many rotations in which that object could be. More specifically, the symmetry set $S(I)$ contains infinitely many rotations. Ideally, an algorithm would provide a way to recover the entire symmetry set, no matter its shape or size. Doing so would give a complete view of the possible orientations for an observed object, which can be very valuable in downstream tasks. Say, for example, we have a robot that is tasked with grabbing a coffee mug. To do so, it has to find the ear of the mug and grab hold of it. If the ear is occluded in the robot's observation, having an idea of the full range of possible orientations allows the robot to reason about where the ear could be (Figure 6.1a). Furthermore, the robot could have a way of performing multiple observations. Either through the use of multiple camera sensors, or by capturing multiple images from a single camera, but at different times. In this case, the information captured by all images would ideally be combined. Orientations that could fit one observation, may be discarded when considering another observation. The symmetry set is a powerful concept in this setting. Suppose we have two observed images I_1 and I_2, then the intersection $S(I_1) \cap S(I_2)$ represents the set of rotations that could explain both images (Figure 6.1b). Conceptually, this allows the robot to shrink its search space progressively with multiple observations.

6.1.1 Binary Classification

In practise, retrieving the entire symmetry set is, however, not feasible. There would not be enough memory in the world to save the infinite number of poses it may contain. Nevertheless, there is a lot of room for improvement with respect to the small number of discrete hypotheses

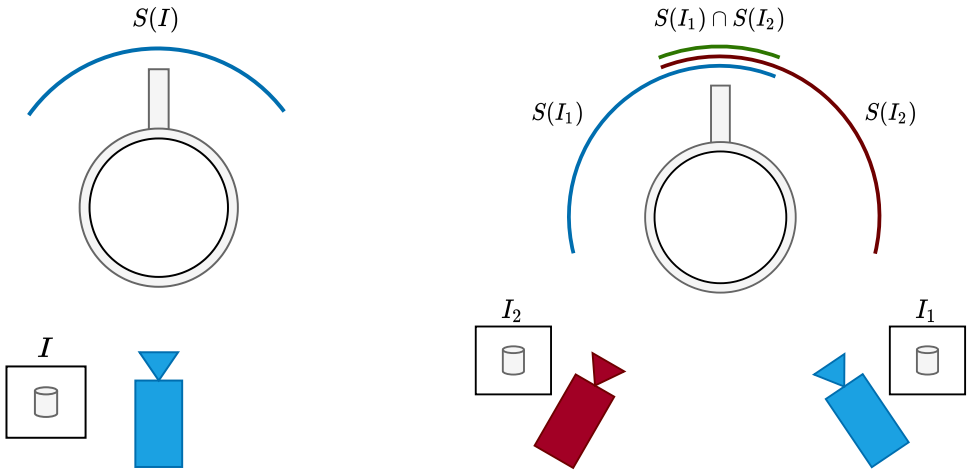


Figure 6.1: When the ear of a mug is occluded from the point of view of a camera, the symmetry set of the observed image will include a continuous range of rotations (a). If multiple observations are available, the intersection of the associated symmetry sets represents the range of poses that fits both observations (b).

given by an MHP model. A powerful alternative to full reconstruction, that is still more expressive than an MHP model, would be an algorithm with which an arbitrary number of hypotheses can be computed. In theory, with an infinite number of diverse hypotheses, the full symmetry set could be reconstructed. In practise, a number can be taken according to computational constraints, or use-case specific needs. This way, the symmetry set can be reconstructed at an arbitrary resolution.

To the end of creating such an algorithm, we can reframe orientation estimation as a binary classification problem. An algorithm in this setting would not output (a discrete set of) rotations directly, but instead classify a given rotation as being part of the symmetry set or not, conditioned on a given image. More formally, we wish to find a classifier

$$\mathcal{C} : SO(3) \times \mathcal{I} \to \{0, 1\} \ , \tag{6.1}$$

which takes a rotation R and an image I as inputs, and classifies whether $R \in S(I)$. Assuming such a classifier $\mathcal{C}$ exists, it is easy to see how it would facilitate the reconstruction of symmetry sets. Note how the classifier accepts the full space of $SO(3)$ as an input domain. Hence, for a given image I, $\mathcal{C}$ can be queried with an arbitrary number of rotations sampled at a continuous resolution. Consequently, $S(I)$ can reconstructed at arbitrary resolution, as far as computational constraints allow.

Furthermore, the existence of $\mathcal{C}$ would also facilitate convenient sampling of intersections of symmetry sets. Let $I_i \in \mathcal{I}$, $1 \leq i \leq N$, be images of the same object, but taken from different view points. Moreover, assume that rotations in the symmetry sets for all images are defined relative to some common frame of reference. Then, as mentioned before, the intersection $\bigcap_{i=1}^{N} S(I_n)$ gives the set of rotations that fit all observations. Note that a given rotation $R \in SO(3)$ is in this set if

and only if it is in each of the separate symmetry sets. More formally,

$$R \in \bigcap_{i=1}^{N} S(I_i) \iff \bigwedge_{i=1}^{N} R \in S(I_i) \ . \tag{6.2}$$

Assuming $\mathcal{C}$ to be a perfectly accurate classifier, we can test whether R is in the symmetry set of an image image I_i by checking $\mathcal{C}(R, I_i) = 1$. Therefore, R is in the intersection if and only if $\bigwedge_{i=1}^{N} \mathcal{C}(R, I_i) = 1$.

The rest of this Chapter details how such a classifier $\mathcal{C}$ can be parameterized by a neural network. The design is based on that of occupancy networks, which will be explained first.

6.2 Review on Occupancy Networks

The idea we propose of a binary classifier used to reconstruct symmetry sets is closely related to recent works in 3D shape reconstruction, as well as the work by Murphy et al. [18]. In 2019, both Mescheder et al [19] and Chen et al. [49] presented similar works, in which 3D shape reconstruction is modeled as a binary classification problem. They both note how a shape can be seen as a set of points in the Euclidean space $\mathbb{R}^3$. Namely, as the set of all points $p \in \mathbb{R}^3$ that are located within the boundaries of the object. In other words, the object *occupies* all such points $p \in \mathbb{R}^3$. Hence, shapes can be described by a so-called *occupancy function*

$$o : \mathbb{R}^3 \to \{0, 1\} \ , \tag{6.3}$$

which classifies whether a given point is occupied by the object or not. Now, given an image $I \in \mathcal{I}$, the shape of the object in I can be output as an occupancy function. While the idea is quite simple, it is not immediately clear how to output such an occupancy function. It is difficult do design a canonical representation without making any assumptions on the shape. Nevertheless, the function should be able to represent any arbitrary shape.

The solution proposed in [19, 49] is to replace the discrete output domain $\{0, 1\}$ of the occupancy function, with the continuous output domain $[0, 1]$. Values in this range are then interpreted as probabilities, such that the output represents the probability that a given point is occupied by the object. By setting a threshold $\mathcal{T}$, a mapping can then be made back to the original classification domain. Every value less than $\mathcal{T}$ is mapped to 0 and every value larger than or equal to $\mathcal{T}$ is mapped to 1. Furthermore, rather than creating a separate mapping from images to occupancy functions, they merge these two stages into a single function

$$f : \mathbb{R}^3 \times \mathcal{I} \to [0, 1] \ . \tag{6.4}$$

In this formulation, there is a single function f that evaluates the occupancy for a given point $p \in \mathbb{R}^3$ with respect to the object in a given image $I \in \mathcal{I}$ directly. Now, f is also in the traditional form of classification problems used in deep learning, and can be approximated by a neural network. Mescheder et al. dubbed such a network an *Occupancy Network*.

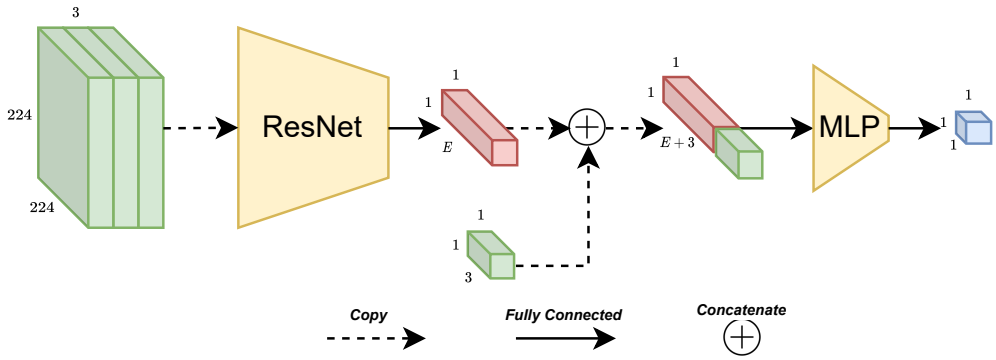


Figure 6.2: A schematic overview of the network architecture used to implement an occupancy network for the purpose of 3D shape reconstruction from images. Images are assumed to be 224 pixels in width and height. A ResNet is used to encode the occupancy function.

6.2.1 Architecture

An occupancy network is implemented according to the architecture shown in Figure 6.2. An image $I \in \mathcal{I}$ is first embedded into a lower dimensional vector of size E. Much like the MHP model in Chapter 4, a pre-trained ResNet model is used to this end by Mescheder et al. Consequently, the image dimensions are fixed to be 224×224, which is the expected input size for ResNet models. The embedding is subsequently concatenated with the query point $p \in \mathbb{R}^3$. The combined vector is then input to a multi-layer perceptron (MLP), which produces the output occupancy probability. The output layer of this MLP uses the *logistic sigmoid* function for its activation, which forces outputs to lie in the range $[0, 1]$. The logistic sigmoid function is commonly denoted by the symbol σ, and defined as

$$\sigma(x) = \frac{e^x}{e^x + 1} \,. \tag{6.5}$$

Note how the extreme output values of 0 and 1 are only reached when the input x tends to $-\infty$ and ∞ respectively, making the practical output range $(0, 1)$. In conclusion, we find a two-stage design, where the first stage extracts features from a given image, embedding it into a low dimensional vector. The second stage then uses this embedding to evaluate an occupancy function at a given point p, conditioned on the image.

This architecture is aimed at providing both versatility and computational efficiency. Having the feature extractor be a separate component makes the design modular. For example, the ResNet model could be swapped out for a PoinNet encoder [50], which would allow the network to take in point clouds as input. Regarding the computational efficiency, we have to note the use case of the occupancy network. In order to reconstruct a polygon mesh from the encoded shape, the occupancy of thousands of points may have to be evaluated. In this two-stage design, the feature extractor is a large and deep network, whereas the MLP is rather compact. Now, since the image stays constant during the mesh reconstruction of a single object, the embedding only has to be computed once. Therefore, the most computationally expensive stage of the network is only inferred once. The occupancies of all subsequent query points can be computed by only evaluating the small MLP model.

6.2.2 Training

As with any binary classification network, occupancy networks require both *positive* and *negative* samples during training. In this case, this implies that training samples should both cover points with an occupancy of 1 and an occupancy of 0 respectively. To this end, the following sampling strategy is applied. A data set is created containing N images. When training on a given image I, a total number of K points are sampled randomly within the bounding volume of the object. For each such point p_i, an associated ground truth occupancy o_i is computed. For convenient notation, let $\mathcal{K}$ be the set of all pairs (p_i, o_i), then the loss for a single image and associated set of samples $\mathcal{K}$ is defined as

$$\mathcal{L}(I, \mathcal{K}) = \sum_{(p_i, o_i) \in \mathcal{K}} H\left(f_\theta(p_i, I), o_i\right) \ , \tag{6.6}$$

where θ are the parameters of the network. This way, a random set of sample points are evaluated for each training image. Furthermore, H is the binary cross-entropy loss function defined as

$$H(o', o) = -o \log(o') - (1 - o) \log(1 - o') \ , \tag{6.7}$$

where o' is the predicted probability of occupancy, and o is the ground truth occupancy. Note that the ground truth o is a binary value in $\{0, 1\}$, as a point is either occupied or not, while the range of o' is continuous. Depending on the value of o, one of the two terms in H is zeroed. Consequently, when $o = 1$, the loss produced by H is minimized when the prediction o' tends to 1. When $o = 0$, the opposite holds, and H is minimized when o' tends to 0. In other words, H is minimized when the predicted occupancy probability approaches the true occupancy. Finally, H is undefined when $o' = 0$ or $o' = 1$, as the logarithm goes to infinity. To prevent this from happening, a small positive value ϵ is usually added to the body of both logarithms.

6.3 Application to Orientation Estimation

With the concept of occupancy networks explained, we revisit the formulation of pose estimation as a binary classification problem. At first glance, the context of 3D shape reconstruction seems completely different from orientation estimation. However, we can draw parallels between the two, where the reconstruction of symmetry sets is analogous to a shape reconstruction problem on the rotation manifold.

6.3.1 Shapes of Symmetry Sets

Analogous to how an object can be seen as a set of 3D points, which together occupy bounded regions in Euclidean space, a symmetry set can be seen as a set of rotations, which together occupy bounded regions on the surface of the rotation manifold. The most notable difference is the input space. In contrast to objects, symmetry sets are constrained to the rotation manifold. When reformulating the occupancy function from Equation 6.3 this need not be a problem, but simply requires a formulation over a different input domain. Hence, an occupancy function describing a

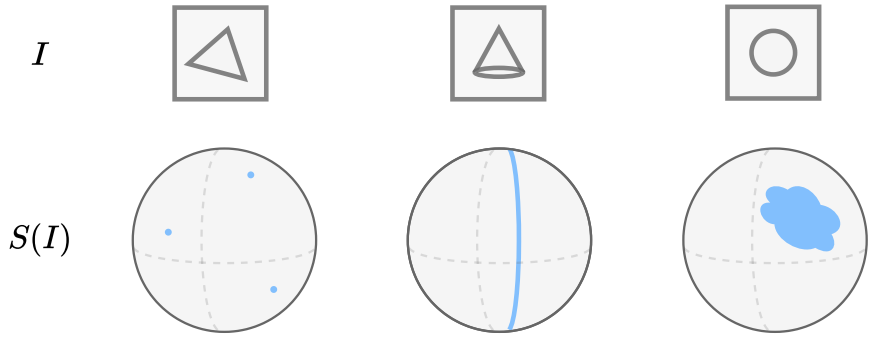


Figure 6.3: Different types of symmetries correspond to differently shaped symmetry sets. Discrete symmetries (left) create symmetry sets consisting in singular points, continuous symmetries around a single axis (middle) trace a continuous loop, and continuous multi-axial symmetries (right) trace a volume. Though symmetry sets are actually defined on the rotation manifold, a simplified 3D interpretation is depicted for clarity.

symmetry set can be formulated as a function

$$o : SO(3) \to \{0, 1\} \ . \tag{6.8}$$

A more impactful difference concerns the shapes of 3D objects and symmetry sets. Where objects typically occupy a single continuous volume in $\mathbb{R}^3$, symmetry sets can have very different shapes. As was briefly mentioned back in Section 4.2, different types of symmetries result in symmetry sets with different properties. Interpretations of this effect are depicted in Figure 6.3. If an image I depicts an object with discrete symmetries, then $S(I)$ will be a finite set. Moreover, the elements of $S(I)$ likely lie apart, thus creating a finite number of disconnected singular points on the rotation manifold. If instead I depicts an object with a continuous symmetry, $S(I)$ will be an infinite set and trace a continuous shape over the manifold. In this case, two notable cases can be distinguished: whether the object is symmetric around a single axis, or around multiple. Take for example a cone, which is symmetric only around a single axis. In this case, $S(I)$ traces a continuous loop around $SO(3)$. Importantly, however, this loop is infinitely thin, causing it to have zero volume. Only when the object has a continuous multi-axial symmetry, will the symmetry set occupy a patch with non-zero volume. Corona et al. [33] proved that such multi-axial symmetries only occur for spheres, making them very rare. Their proof only pertains to global symmetries, however, and does not cover occlusion-induced symmetries. Nevertheless, we conjecture that multi-axial symmetries can only be induced if an object appears like a perfect sphere under some occlusion.

Since discrete and single-axis symmetries are very common, the shapes of symmetry sets often consist of thin structures. For such structures to be accurately modeled, the occupancy function must produce sharp peaks. However, recent analyses [51] has shown that the high frequencies required to produce these peaks are difficult to model using an MLP. This will be an important consideration when designing an occupancy network for modeling symmetry sets.

6.3.2 Network Design

With the analogy to 3D shape reconstruction clear, we can apply the concept of an occupancy network to orientation estimation. The basic idea stays virtually unchanged. Conceptually, we again find a two-stage process, where the first stage extracts features from a given image, embedding it into a low dimensional vector. The second stage uses the embedding, in combination with a given rotation, to evaluate an occupancy function on $SO(3)$ (Equation 6.8), conditioned on the image. Like in Equation 6.4, these two stages are combined into a single function formulation as

$$f_\theta : SO(3) \times \mathcal{I} \to [0, 1] \ , \tag{6.9}$$

parameterized by a neural network with parameters θ. Note that the output domain is again changed from $\{0, 1\}$ to $[0, 1]$ to fit the traditional formulation of a binary classification problem. Outputs can be interpreted as probabilities, giving the probability that a given rotation $R \in SO(3)$ is in the symmetry set $S(I)$ of a given image $I \in \mathcal{I}$.

Rotation Representations

In this formulation, the only thing that has changed with respect to a standard occupancy function is the input domain. More specifically, the new formulation operates on rotations instead of 3D Euclidean coordinates. In practise, this requires picking a suitable representation for rotations to be used as an input format for the neural network. In contrast to the simple 3D vectors describing points from $\mathbb{R}^3$, there are several options for representing rotations. Candidates include Euler angles, rotation vectors, unit quaternions, and rotation matrices. Respectively, these can be represented as 3D vectors, 3D vectors, normalized 4D vectors, and orthogonal 3×3 matrices.

As was discussed in Section 2.1, each representation has its own characteristics. An important benefit of Euler angles and rotation vectors is that every possible 3D vector corresponds to a valid rotation. With quaternions and rotation matrices however, only normalized and orthogonal values are proper rotations respectively. As a consequence, decision boundaries drawn through the input space can cross values that do not actually belong to $SO(3)$. Nevertheless, unit quaternions and rotation matrices benefit from their accurate representation of the topology of the rotation manifold. Notably, special care ought to be taken to deal with the antipodal symmetry of unit quaternions. Two antipodal unit quaternions represent the same rotation, and, therefore, should result in the same output. This problem can be avoided by projecting all unit quaternions to a single half of the unit hypersphere before using them as input to the network, thus creating a canonical value for antipodal quaternions.

In a similar network design, Murphy et al. [18] tested all the representations mentioned above. Following their conclusions, we pick rotation matrices as the representation for input rotations, thus changing the formulation in Equation 6.9 to

$$f_\theta : \mathbb{R}^9 \times \mathcal{I} \to [0, 1] \ . \tag{6.10}$$

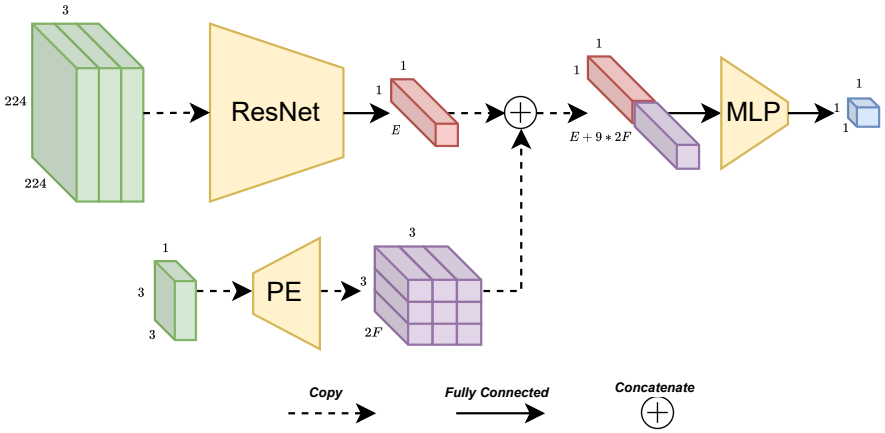


Figure 6.4: A schematic overview of the occupancy network used to reconstruct symmetry sets.

Positional Encoding

To address the issue of thin shapes common in symmetry sets, we apply a *Positional Encoding* (PE) to all input rotations before feeding them to the MLP. As mentioned in Section 6.3.1, MLP networks have been shown to have difficulties learning high frequency functions, which have sharp local fluctuations [51]. Nevertheless, in order to properly represent thin symmetry sets, we require our MLP to produce sharp peaks in the occupancy function, requiring high frequency fluctuations.

Recent work has addressed this issue by manually mapping input values to vectors in a higher dimensional space using high frequency functions. The resulting vector is then used as input to the network instead. Mildenhall et al. [52] first showed how this technique helped in reconstructing fine details in shapes. Murphy et al. [18] have subsequently shown its successful application to input rotations. The most popular mapping applies a function $\gamma : \mathbb{R} \to \mathbb{R}^{2F}$, defined as

$$\gamma(x) = \left(\sin\left(2^0\pi x\right), \cos\left(2^0\pi x\right), ..., \sin\left(2^{F-1}\pi x\right), \cos\left(2^{F-1}\pi x\right)\right) \ , \tag{6.11}$$

where $x \in \mathbb{R}$, and F is the number of frequencies. Higher values of F exponentially increase the frequency with which inputs are embedded. Furthermore, note that γ is applied to individual input values. In our use-case, that means γ is applied separately to each of the 9 values making up an input rotation matrix. Therefore, we effectively create a mapping from $\mathbb{R}^9$ to $\mathbb{R}^{9\cdot 2F}$.

Rahaman et al. [51] give a detailed mathematical explanation as to why the use of γ helps MLP networks to learn high frequency functions. In short, their conclusion is that a high frequency function on the input space of γ ($\mathbb{R}^9$ in our case) can be expressed as a low frequency function on the output space of γ ($\mathbb{R}^{9\cdot 2F}$ in our case) which are easier to learn for an MLP. A perhaps more intuitive explanation as to why the positional encoding works is as follows. Input values that lie closely together in the original input space $\mathbb{R}^9$ are mapped to vectors that lie farther apart in $\mathbb{R}^{9\cdot 2F}$, due to the high frequencies introduced in γ. Hence, they are more easily distinguished, allowing the network to create sharper decision boundaries.

Final Design

Putting everything together, we arrive at the architecture described in Figure 6.4. Images are first embedded into a lower dimensional vector of size E using a pre-trained ResNet model. The embedding is subsequently concatenated with the positional encoding of a query rotation, and fed to an MLP model. Note that the positional encoding contains $9 \cdot 2F$ values, since Equation 6.11 contains both a sine and a cosine for each frequency. This makes the total combined input size for the MLP model $E + 9 \cdot 2F$. Finally, the output of the MLP model is put through a Sigmoid function, which ensures that its value lies in the domain $[0, 1]$.

6.3.3 Training Strategy

When applying the training process of a traditional occupancy network to the proposed network, a problem becomes apparent. As was explained in Section 6.2.2, both *positive* and *negative* samples are required to train a classifier. In the occupancy network, these samples are provided by randomly sampling points inside the bounding volume of an object and testing whether they lie inside or outside the object's shape. This is possible because the shape of the object is known through a provided CAD model. The problem appears when we try to do the same for symmetry sets. Random samples cannot be checked to lie inside or outside the symmetry set, because no ground truth representation of the symmetry set is assumed to be available. Unlike object shapes, which are relatively easy to recreate using 3D modeling tools, symmetry sets are too difficult to annotate. In the presence of self-occlusion, for example, the exact shapes of symmetry sets can vary dramatically from one image to the next. It is in fact this complex relation between images and symmetry sets that motivates the design of this network in the first place.

The only data that is assumed to be available during training consists of image-rotation pairs, where each image is annotated only with the corresponding ground truth pose. In terms of a classification problem, this means only a single, positive sample is available for each image. More formally, such a data set $\mathcal{D}$ can be described as

$$\mathcal{D} = \{(I_i, R_i) : 1 \leq i \leq N, I_i \in \mathcal{I}, R_i \in SO(3)\} \ , \tag{6.12}$$

where N denotes the number of pairs in the data set. With only this information, training the proposed network like an occupancy network seems impossible. Nevertheless, we find that by restricting the supported types of symmetries to only discrete and continuous single-axis, a training strategy for the proposed network can be formulated using only image-rotation pairs.

Restricting to Discrete and Single-Axis Symmetries

The proposed training strategy relies on a particular property of discrete and single-axis symmetries, namely that they have zero volume on the rotation manifold. As a consequence, an image I that exhibits only a discrete or continuous single-axis symmetries, or a combination thereof, will produce a symmetry set with zero volume. Hence, a random rotation R_i sampled uniformly from $SO(3)$ falls inside $S(I)$ with probability zero. It can, therefore, be assumed that for any such R_i it holds that $R_i \notin S(I)$.

With this key observation, we can turn back to the training process of occupancy networks.

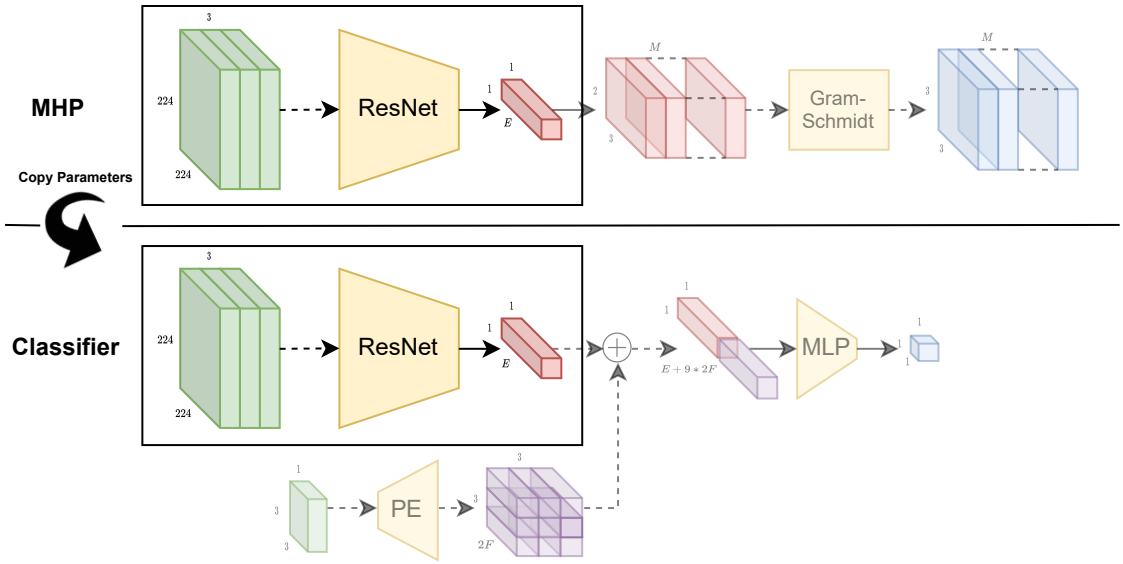


Figure 6.5: Parameters from a trained MHP network can be used as pre-trained weights for the first stage of a binary classification model.

It is no longer necessary to have annotated ground truth symmetry sets, now that any randomly sampled rotation is known to fall outside the symmetry set. When recreating the loss from Equation 6.6, this implies that each of the K random samples, for each image in a batch, has an associated occupancy of 0. This is also problematic, since the network would only be given *negative* samples. We address this as follows. Rather than using only randomly sampled rotations, we additionally evaluate the network at the ground truth rotation. Given an image-pose pair $(I, R) \in \mathcal{D}$, this means R is manually added along side the K randomly sampled rotations, in order to introduce a *positive* sample. More formally, we define the loss for a single image-rotation pair to be

$$\mathcal{L}(I, R) = H\left(f_\theta(I, R), 1\right) + \sum_{i=1}^{K} H\left(f_\theta(I, R_i), 0\right) \ , \tag{6.13}$$

where H refers to the binary cross-entropy loss function mentioned in Equation 6.7

$$H(o', o) = -o \log(o') - (1 - o) \log(1 - o') \ . \tag{6.14}$$

Note that $o = 1$ results in the second term becoming zero, and that $o = 0$ results in first term becoming zero. Hence, Equation 6.13 can be rewritten to

$$\mathcal{L}(I, R) = -\log\left(f_\theta(I, R)\right) + \sum_{i=1}^{K} -\log\left(1 - f_\theta(I, R_i)\right) \ . \tag{6.15}$$

Pre-training the First Stage

Initial experiments using the loss from Equation 6.15 showed inconsistent results. During training, the network has a tendency get stuck in a local optimum. More specifically, it gets stuck outputting

0 for all inputs. Considering Equation 6.15, this strategy achieves an optimal score on all K negative samples, at the expense of very poor score on the ground truth rotation.

Further experiments showed that pre-training the first stage of the proposed network as a multiple-hypothesis prediction model helped to increase robustness (Figure 6.5). A quantitative evaluation to substantiate this claim is presented in Section 7.4.1. Recall the MHP architecture detailed in Section 4.2.2. The first stage of the binary classification network is identical to that of an MHP network. The results presented in Chapter 5 show how MHP models are able to predict diverse sets of hypotheses covering symmetry sets. To achieve this, the parameters of the underlying neural network are able to extract features that help generating these hypotheses. The intuition behind copying these parameters to a binary classification network, is that the same features will be useful for classification. By copying them, the training process is preconditioned, helping to avoid the aforementioned local minimum.

Chapter 7

Experiments with Binary Classifiers

This Chapter details experiments that evaluate the newly proposed binary classification network for orientation estimation (Chapter 6). The structure of the chapter is similar to that of to Chapter 5. First, Section 7.1 covers the data set on which the new design is evaluated. Subsequently, Section 7.2 introduces terminology related to the evaluation of binary classifiers, and describes the metrics used in the final evaluation. Section 7.3 covers implementation details. Finally, results are presented and discussed in Section 7.4.

7.1 Data Set

The requirements for a data set in this evaluation are mostly the same as those for multiple-hypothesis prediction evaluation (Chapter 5). That is, the data set should cover objects with different types of symmetries, and should only pertain to orientation estimation. Furthermore, each image in the data set should be annotated with a single ground truth pose, but also its full ground truth symmetry set. Since the proposed model is specifically designed to reconstruct symmetry sets, these ground truth symmetry sets are particularly important to have available. Without ground truth symmetry sets, the quality of a model can not properly be evaluated.

In contrast to multiple-hypothesis models, the proposed binary classification model makes an additional assumption that further constrains the data set. Namely, that all objects can only exhibit discrete symmetries or single-axis symmetries. In practise, this excludes spheres, or objects that can appear spherical under occlusion. Since such objects are quite rare, this additional requirement is not particularly restrictive.

Considering these requirements, we again choose the SYMSOL data set with custom symmetry set annotations as introduced in Section 5.1. Though the original SYMSOL II data set contains the marked sphere object, which can appear like a white sphere under occlusion, it is not contained in custom annotated version. All other objects only exhibit discrete or single-axis symmetries, making the data set suitable for use in this evaluation. Furthermore, the objects appear in a simplistic setting: without noise and under constant lighting conditions. Feature extraction, in

this setting, is no problem for a contemporary convolutional neural network like ResNet. Therefore, the bottleneck in training the proposed binary classification models will not be the ability of the backbone to perform feature extraction, but rather the full model's ability to learn which parts of the rotation manifold belong to the symmetry set.

7.2 Evaluation Metrics

The goal of training the proposed model, is to classify whether a given rotation is part of the symmetry set of a given image. Hence, each model should be evaluated based on its performance as a classifier. Traditionally, this performance is best captured by measuring *recall* and *precision*.

7.2.1 Classification Terminology

As was explained earlier, the proposed model predicts whether a given rotation is inside the symmetry set of a given image or not. This makes the model a function over the combined space of rotations and images

$$f_\theta : SO(3) \times \mathcal{I} \to [0, 1] \ . \tag{7.1}$$

However, when considering only a single image $I \in \mathcal{I}$, we are left with a classification problem solely over $SO(3)$. In this context, the network predicts whether a given rotation is inside the symmetry set $S(I)$. Therefore, conceptually, every image is associated with its own instance of a classification problem over just the space of $SO(3)$. For a fixed image I, the model becomes a function

$$f_\theta^I : SO(3) \to [0, 1] \ . \tag{7.2}$$

For the rest of this evaluation, we measure the performance of the proposed model on a per-image basis, and compute average metrics over all images in the data set.

Since the proposed model predicts two classes, we can evaluate its performance using established metrics for evaluating binary classification models. To this end, it is productive to formulate our problem in the terminology used in the related literature. Consider the model as a classifier

$$\mathcal{C}_\theta^I : SO(3) \to \{0, 1\} \ , \tag{7.3}$$

for a fixed image $I \in \mathcal{I}$. This model is said to classify *samples* from $SO(3)$ into two classes: the *positive* class (1) and the *negative* class (0). Here, a sample $R \in SO(3)$ is said to be in the positive class when $R \in S(I)$, and in the negative class when $R \notin S(I)$. Note, however, that our network does not predict this class directly, but rather rather outputs the probability of a sample being positive. To convert this output probability to a binary prediction, we set a probability threshold $\mathcal{T}$, such that

$$\mathcal{C}_\theta^I(R) = \begin{cases} 1 & \textbf{if } f_\theta^I(R) > \mathcal{T} \\ 0 & \textbf{if } f_\theta^I(R) \leq \mathcal{T} \end{cases} \tag{7.4}$$

Now, to evaluate a model's performance on an image, we take a set of samples for which their classes are known. In other words, for every sample, it is known whether it belongs to the positive class or the negative class. When these samples are input to the classifier, the model

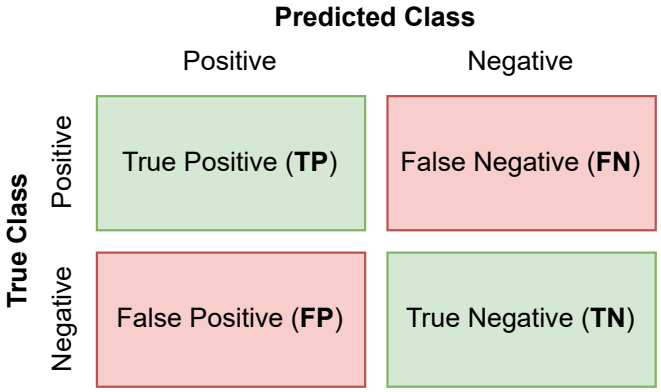


Figure 7.1: The confusion matrix categorizes binary predictions into four categories, depending on the predicted class of a sample and the predicted class of a sample.

makes predictions on whether a sample is positive or negative. Ideally, the predictions agree with the truth. However, in practice, trained classifiers are usually not perfect, and the predictions for some samples will be wrong. The relationship between prediction and truth is intuitively captured in a confusion matrix (Figure 7.1). According to the confusion matrix, a prediction can fall into one of four categories as follows.

- **True Positive (TP)**: a *positive* sample is correctly predicted to be *positive*.
- **False Positive (FP)**: a *negative* sample is wrongly predicted to be *positive*.
- **False Negative (FN)**: a *positive* sample is wrongly predicted to be *negative*.
- **True Negative (TN)**: a *negative* sample is correctly predicted to be *negative*.

Counting the number of predictions that fall into each category gives valuable data regarding the quality of a classification model. For more details on this topic, the reader is referred to [53]. Later in this section, it is explained how this data is used to construct informative metrics, which are used to quantitatively evaluate model performance. Firstly, however, we discuss how a set of samples is generated for every image.

7.2.2 Generating Samples

For every image I, a set of sample rotations is required that covers both the positive and negative class. Finding positive samples, that lie inside the symmetry set $S(I)$, is simple. Note that the full (discretized) symmetry set is available for every image in our annotated version of the SYMSOL data set (Section 5.1). All elements of these annotated symmetry sets can be directly used as positive samples for evaluation.

Negative samples can not be deduced directly from the annotated data. However, since our classification model (and data set) are restricted to objects with discrete symmetries or single-axis continuous symmetries (Section 6.3.3), we can assume that all images have symmetry sets with zero volume. Consequently, any rotation sampled uniformly random from $SO(3)$ will fall outside

the symmetry set with probability 1, and will therefore be a negative sample. With this knowledge, an arbitrary number of negative samples can be generated for every image.

However, for the sake of reproducibility of evaluation results, it would be desirable to use a deterministic set of samples, rather than random one. Moreover, the local density of randomly generated samples inevitably varies across the rotation manifold. Therefore, a peak in the output of the network can coincide with a different number of samples depending on this random local density. Consequently, two equally sized peaks may incur a significantly different penalty. For these reasons, rather than generating a random set of samples, we instead follow Murphy et al. [18] and adopt a technique for generating equivolumetric grids on $SO(3)$ developed by Yershova et al. [23]. This technique generates a deterministic grid of rotations on $SO(3)$, such that all grid cells have the same volume. Moreover, the grid can be generated at multiple hierarchical levels, each level containing exponentially more points. In this evaluation, we use the fourth hierarchical level, which is composed of 294 thousand rotations. This grid will be referred to by the symbol $\mathcal{G}$ in the rest of this section.

Note that the rotations in $\mathcal{G}$ are not distributed uniformly random. In contrast, they are placed deterministically, and in a regular pattern with uniform density on $SO(3)$. Nevertheless, for the rotations in $\mathcal{G}$ to be considered proper negative samples, they must fall outside the symmetry set of every image. Since the images in the SYMSOL dataset were generated using rotations that are themselves uniformly randomly sampled, the relative orientation of $\mathcal{G}$ with respect to any such rotation is also random. In turn, the probability of the symmetry set of any image aligning with $\mathcal{G}$ is still zero. Hence, all rotations in $\mathcal{G}$ are still guaranteed to be proper negative samples, as they fall inside any symmetry set from SYMSOL with probability zero.

7.2.3 Recall

The recall of a binary classifier measures its ability to correctly identify *positive samples*. More specifically, it is defined to be the proportion of positive samples that are correctly classified by the model. This value is calculated as

$$\texttt{recall} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FN}} = \frac{\texttt{TP}}{|S(I)|} \ . \tag{7.5}$$

Ideally, the recall is 1, in which case the classifier correctly classifies all positive samples. In context, this would mean that the entire symmetry set is found.

7.2.4 Precision

Just the recall alone does paint not a complete picture of model performance. Even when a classifier achieves a recall score of 1, it can still give undesirable results. This is because recall does not depend the number of false positives or true negatives. Consider a classifier that classifies every sample as positive. This model would not generate any false negatives, because never predicts the negative class. Consequently, its recall score will be 1. Nevertheless, it also wrongly classifies all negative samples, making it useless in practise.

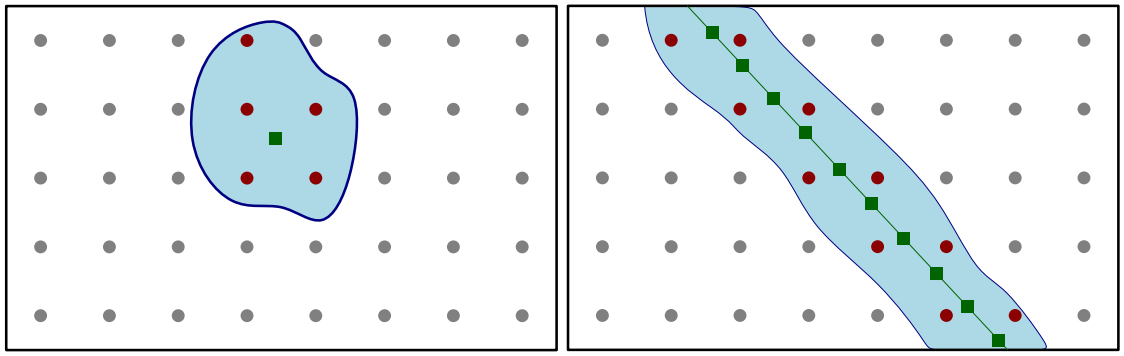
An additional metric is needed that penalizes false positive predictions. To this end, we use *precision*, which is defined as the proportion of samples that are predicted positive that are truly

positive. This value is calculated as

$$\texttt{precision} = \frac{\texttt{TP}}{\texttt{TP} + \texttt{FP}} \,. \tag{7.6}$$

At a precision score of 1, a model generates no false positive predictions. A special case occurs when $\texttt{TP} + \texttt{FP} = 0$, which happens when the network does not predict any samples to be positive. In this case, we say the precision is 1, since no false positives have been generated, and the recall would be 0.

To see how precision relates to the classification of symmetry sets, see Figure 7.2. For a given image I, the classifier $\mathcal{C}_\theta^I$ predicts certain regions of $SO(3)$ as either positive or negative. The boundary between such regions is called the *decision boundary*, and depends on the threshold $\mathcal{T}$. Samples from $\mathcal{G}$ that fall inside the decision boundary are predicted as false positives. Intuitively, the tighter the decision boundary fits around the symmetry set, the fewer false positives will be predicted, and the higher the precision will be.



(a) Discrete (b) Continuous

Figure 7.2: An illustration showing how the decision boundary relates to precision for both discrete and continuous symmetries. The enclosed blue area represents the region of $SO(3)$ predicted as positive. Green squares represent samples from $S(I)$ that are predicted positive. Moreover, in (b), the green line represents the true continuous symmetry set. Circles represent the grid $\mathcal{G}$, where gray circles are true negative predictions and red circles are false positive predictions.

7.2.5 A Summary Metric

Since neither recall nor precision are enough to evaluate performance on their own, they are best used together. Nevertheless, it is preferable to use only use a single metric, as this would make it easier to compare different models. Such a metric should summarize both recall and precision into a single value. Traditionally, the *accuracy* score fills this role, where the accuracy defines the proportion of all samples that are predicted correctly. More formally, it is calculated as

$$\texttt{accuracy} = \frac{\texttt{TP} + \texttt{TN}}{\texttt{TP} + \texttt{FP} + \texttt{FN} + \texttt{TN}} = \frac{\texttt{TP} + \texttt{TN}}{|S(I)| + |\mathcal{G}|} \,. \tag{7.7}$$

The accuracy score can, however, be problematic. When the samples are heavily imbalanced, meaning that one class is represented much more frequently than the other, the accuracy becomes difficult to interpret. If 95% of samples are negative, than an accuracy of 0.95 can be achieved by

predicting all samples as negative. This score is deceptively high, while the model itself is useless.

The samples used in this evaluation are also very imbalanced. Even the largest symmetry sets in the data set only consist of 720 rotations. The grid $\mathcal{G}$, on the other hand, has almost 300 thousand rotations. Therefore, there are many times fewer positive samples than negative samples.

As a replacement for the accuracy, we instead use the popular *F-measure*, which is defined as the harmonic mean between recall and precision. This metric does not depend on the number of true negative predictions, which can be become very high due to the large imbalance. It is computed as

$$\texttt{F-measure} = 2 \cdot \frac{\texttt{recall} \cdot \texttt{precision}}{\texttt{recall} + \texttt{precision}} \,. \tag{7.8}$$

7.3 Implementation Details

The binary classification models were implemented in PyTorch, according to the network design outlined in Section 6.3.2. The first stage of the network was implemented like that of the MHP networks explained in Section 5.3. This way, the weights of a pre-trained MHP model can be easily copied into the binary classification network. To reiterate, the 18-layer version of ResNet is used in the first stage. Red, green, and blue channels of images are normalized using mean 0.485, 0.456, and 0.406 and standard deviation 0.229, 0.224, and 0.225 respectively, before being input to the model, with the purpose of being compatible with pre-trained versions of ResNet available in PyTorch[1]. The output of this ResNet model is processed into an embedding by a fully connected layer with ReLU activation. The embedding size E is again fixed at 256. Based on empirical observations, three frequencies were used in the positional encoding, such that $F = 3$.

The second stage MLP network was implemented with two hidden layers, each with 256 neurons and ReLU activations. The output layer consists of a single neuron with logistic sigmoid activation (Equation 6.5), which forces the output to lie in the range $[0, 1]$.

All networks were trained for a total of 30 epochs. Training for additional epochs was not found to significantly improve convergence. The batch size was set at 32, with larger batch sizes being found to result in worse F-measure scores. Network parameters were, again, optimized using the ADAM optimizer. The learning rate starts at $1e^{-3}$, and is lowered to $1e^{-4}$ after epoch 15, and then to $1e^{-5}$ after epoch 25.

Finally, it was found to be beneficial to freeze the parameters of the ResNet model, as well as the layer connecting it to the embedding, at the start of training. At this point, the weights of the second stage MLP are still set to their randomly initialized values. When using a pre-trained MHP network in the first stage (Figure 6.5), its weights are already meaningful. Therefore, the idea is to first let the second stage converge while keeping the first stage constant. After some time, the weights of the first stage are unfrozen, and the network is trained end-to-end. Here, we choose to unfreeze after the 4^{th} epoch.

[1] `https://pytorch.org/hub/pytorch_vision_resnet/`

7.4 Evaluation

The evaluation of the proposed binary classification model will be separated into two different experiments. The first is an ablation study, in which we motivate the use of a positional encoding described in Section 6.3.2, as well as the use of a pre-trained MHP model in the first stage (Section 6.3.3). The second studies how the number of samples K used during training, influences the performance of the final model.

7.4.1 Ablation

The goal of this ablation study is to test whether the addition of a positional encoding and pre-training of the first stage actually increase performance. To this end, we train the proposed model in various configurations.

As a baseline, we use a model with positional encoding, and use the weights from the MHP model with the highest performance on recall (Section 5.4) in the first stage. This baseline is compared against three other models, that each change one component with respect to the baseline. The first uses no positional encoding. The second uses no MHP pre-training, but instead uses the default weights for ResNet-18 pre-trained on the ImangeNet classification data set. Lastly, the third uses weights from the MHP model with the highest performance on precision.

When selecting the MHP model with best precision or recall, the model is chosen with the lowest mean error. In the event of a tie, the model is chosen with the lowest mean error on the opposite metric. Note that for the icosahedron, cone, and cylinder, the MHP model performing best on precision and recall is the same. All models are trained with $K = 256$ samples during training. F-measure scores for all models are shown in Table 7.1

	SYMSOL I					SYMSOL II	
	tet	cube	icosa	cone	cyl	tetX	cylO
No Positional Encoding	0.000	0.169	0.000	0.964	0.974	0.219	0.560
No MHP Pre-training	0.565	0.755*	0.696	0.949*	0.971*	0.240*	0.560*
Best Precision	0.551	0.804	-	-	-	0.381	0.572
Baseline (Best Recall)	0.619	0.837	0.670	0.959	0.978	0.408	0.603

Table 7.1: Mean F-measure scores for different ablations of the binary classification model. The thresholds was set at $\mathcal{T} = 0.8$ for all models. The best performing model on each object has a dark gray colored cell. Higher is better (↑). (*) Learning rate was set to $1e^{-4}$ already after epoch 4, otherwise the model would not train.

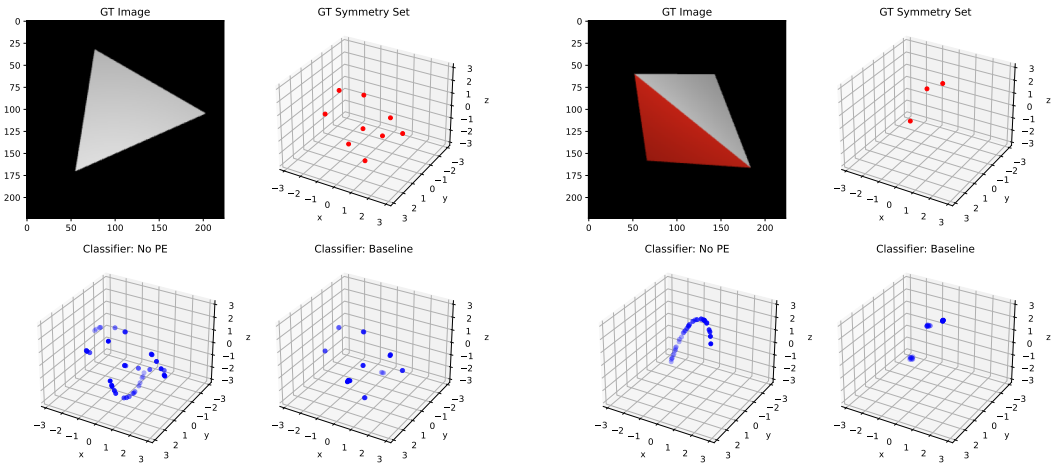
The baseline model consistently shows good performance with respect to the other models, achieving top scores for all but the icosahedron and cone. In general, differences in scores are largest on objects with discrete symmetries. Appendix B contains example predictions for both SYMSOL I and SYMSOL II data sets. Several observation can be made from the data, regarding the individual network components.

Positional Encoding

Observations regarding the positional encoding mostly agree with those of Murphy et al. [18]. For objects with continuous symmetries, specifically the cone and cylinder from SYMSOL I, the posi-

tional encoding has an almost negligible impact on performance. On the cone object, performance even slightly improves over the baseline model. However, whereas Murphy et al. observed a slight decrease in performance on all objects with continuous symmetries, our data does not support this conclusion in a general sense.

In contrast, for object with discrete symmetries, the benefits are evident. Models for the tetrahedron, as well as the icosahedron from SYMSOL I, failed to train at all without the positional encoding. More specifically, they fail to converge to a meaningful solution, getting stuck outputting 0 for all inputs. Results for the cube and marked tetrahedron did not get stuck in the same way, but perform significantly worse than the baseline. Figure 7.3 shows how the model without positional encoding is unable to produce a tight decision boundary around the symmetry set, in contrast to the baseline.



(a) (b)

Figure 7.3: π-Ball plots showing the difference between using positional encoding (PE) and not using positional encoding. Classifiers are evaluated on images from the marked tetrahedron data set from SYMSOL II. For each classifier, only positive predictions are shown.

Pre-Training

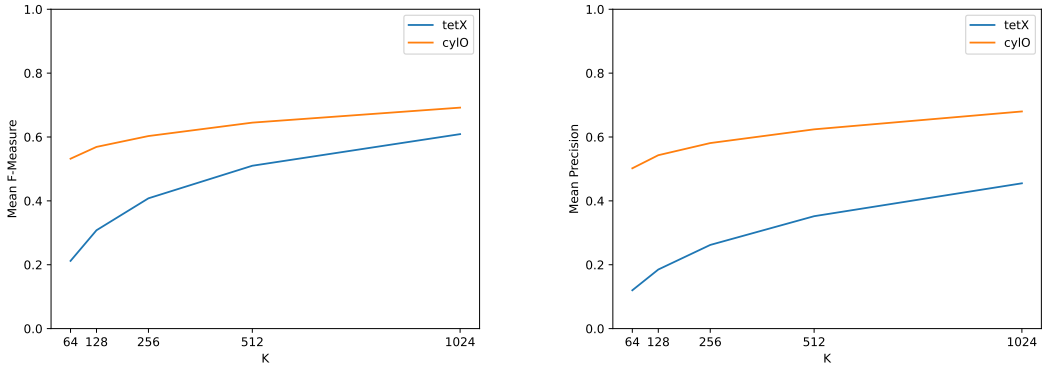
Like with the use of a positional encoding, differences in performance between various types of pre-training mostly show for discrete symmetries. Unlike the models without positional encoding, all models with no MHP pre-training achieve F-measure scores that lie reasonably close to the baseline. However, for several such models, the training process had to be altered for the networks to train at all. Similar to the configuration without positional encoding, these networks got stuck outputting 0 for all inputs. In general, training networks without MHP pre-training was found to be a delicate process. Robustness of the training process greatly improved when adding the pre-training step. Furthermore, the use of the MHP model with highest recall (baseline) consistently outperforms the model with highest precision.

7.4.2 Training Sample Sizes

Another important variable in the training process is the number of samples K used during training. To re-iterate, for a given image-pose pair (I, R), the loss used to train the network f_θ is

$$\mathcal{L}(I, R) = -\log\left(f_\theta(I, R)\right) + \sum_{j=1}^{K} -\log\left(1 - f_\theta(I, R_j)\right) \ , \tag{7.9}$$

where K specifies the number of randomly sampled rotations used as negative samples. The higher K, the more densely these negative samples cover $SO(3)$. To study the impact of K on performance, we train the baseline model from the previous section with different values of K on the SYMSOL II data set. Specifically, the values 64, 128, 256, 512, and 1024 are used. Figure 7.4 shows how the mean F-measure changes.



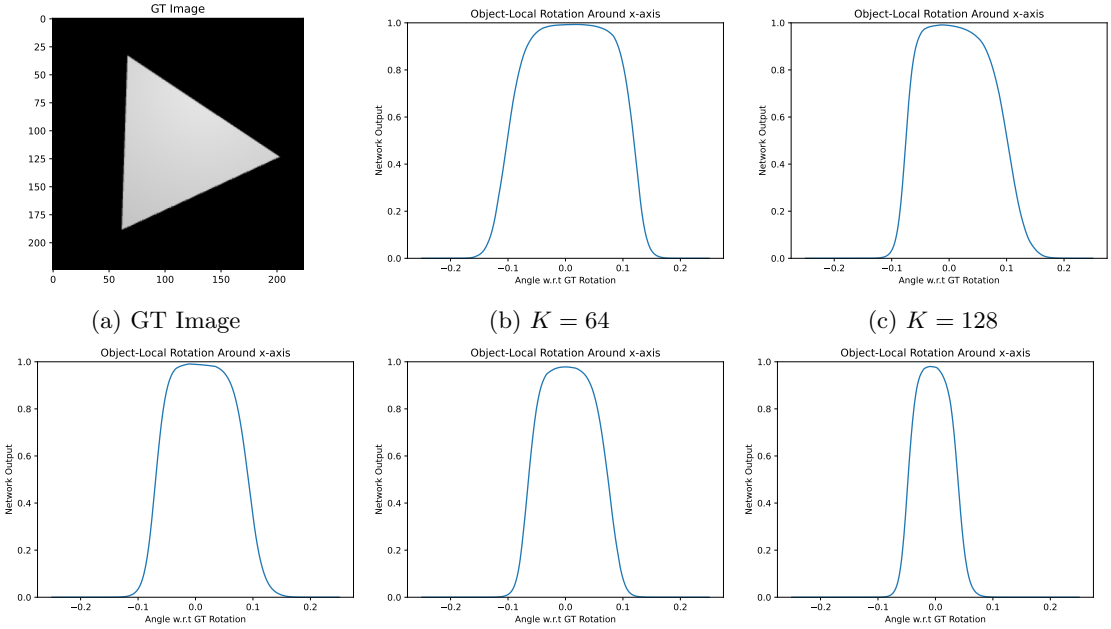
(a) F-measure (b) Precision

Figure 7.4: Plots of mean F-measure and mean precision scores for the baseline model over the sample size K used during training. Results are shown for the SYMSOL II data set, where tetX and cylO refer to the marked tetrahedron and marked cylinder respectively. A threshold of $\mathcal{T} = 0.8$ is used in for all models.

The plots show how increasing the sample size significantly improves F-measure scores. These improved scores can be attributed mostly to increases in precision (Figure 7.4b). Recall stays mostly the same, with a marginal decrease for higher values of K. Between the marked cylinder and marked tetrahedron, the latter enjoys the largest increase in scores. This is consistent with earlier results, where objects with discrete symmetries were found to be challenging for the classifier to deal with.

Deeper investigation gives an explanation for improved scores on the marked tetrahedron. In order to precisely classify its discrete symmetries, the network must produce very sharp peaks in its output. This way, the decision boundary fits tightly around the individual symmetries, thus increasing precision. With a denser sampling of rotations during training, it becomes more likely for these samples to appear in close proximity to the symmetries of the tetrahedron. Consequently, the network is incentivized to tighten its decision boundary around these symmetries, as to correctly distinguish them from the random negative samples during training. Therefore, it is intuitive that precision increases for higher values of K.

Figure 7.5 illustrates this effect, by zooming in on one of the peaks generated in the output of the network. A one-dimensional slice of rotations is taken along the object-local x-axis, and the output of the network is computed at each rotation. These plots give a clear example of how larger values of K result in sharper output peaks, thus increasing precision.



(a) GT Image (b) $K = 64$ (c) $K = 128$

(d) $K = 256$ (e) $K = 512$ (f) $K = 1024$

Figure 7.5: Plots zooming in on a single output peak produced by various binary classification models on an image of the marked tetrahedron. Each model is trained with a different number of training samples K. The output is shown over a one-dimensional slice of rotations taken along the object local x-axis, centered on the ground truth rotation. Relative angles on the x-axis are shown in radians.

7.4.3 Summary

The proposed binary classification network is able to faithfully reconstruct symmetry sets for images of the SYMSOL data set. This is true for both the simplistic symmetry sets created by globally symmetric objects, as well as more complicated sets created by nearly symmetric objects.

An ablation study shows how the addition of a positional encoding, as well as a pre-trained first stage, help increase (1) the robustness of the training process and (2) the F-measure scores of the model. The positional encoding is found to be particularly influential, allowing networks to produce tight decision boundaries around discrete symmetric rotations.

Finally, increasing the sample size parameter K during training is shown to further increase precision. Increasing the number K incentivizes the network to produce sharper peaks in its output. The benefits of this are most significant for discrete symmetries, where the decision boundary is preferably as tight as possible.

Chapter 8

Conclusions

This chapter summarizes the most important findings of this thesis, and presents recommendations for future work.

8.1 Summary

In this thesis, we have addressed the issues posed by rotational symmetries in orientation estimation. Existing literature in pose estimation either uses manual canonicalization of symmetric poses, or symmetry-aware loss functions based on object CAD models, to predict a single correct pose. Other work treats ambiguity as a cause of uncertainty, predicting distributions over poses. The former approach gives no information on the set of possibly correct poses under ambiguity, called the symmetry set, and does not scale to symmetries induced by occlusions. The latter often requires parameterization and computationally expensive normalization to obtain a complex distribution on $SO(3)$.

Ground truth symmetry set annotations for every image are required to evaluate orientation estimation algorithms on symmetric objects and nearly-symmetric objects. To this end, in Chapter 5, we have introduced the first data set with annotated symmetry sets covering nearly-symmetric objects under self-occlusion. This data set is based on the SYMSOL II data set introduced in earlier work. Known patterns in the symmetries are exploited to generate candidate rotations for the symmetry set of every image. A methodology for accepting or rejecting candidates is presented that uses rendered segmentation maps to check images for visible markings.

In Chapter 4, we have detailed the application of multiple-hypothesis prediction to the task of orientation estimation. Subsequently, in Chapter 5, three meta loss functions, as well as two embedded loss functions, for multiple-hypothesis prediction models have been evaluated. Results show how well sets of hypotheses, output by various combinations, represent symmetry sets, where performance is measured using quantitative measures of precision and recall. Our findings identify properties of the Von Mises loss function that favorably impact the precision of hypotheses when compared to the traditional geodesic loss. Moreover, we have found performance to vary the most on objects with discrete symmetries, and saw best performance using the evolving winner takes all training strategy.

Finally, in Chapter 6, we have modeled orientation estimation as a binary classification prob-

lem on the combined space of images and $SO(3)$. Our solution is based on existing work in implicit shape reconstruction and pose distribution estimation, but requires a restriction to only discrete and single-axis symmetries. The resulting network is able to learn accurate decision boundaries, separating symmetry sets from the rest of $SO(3)$ at continuous resolution. In Chapter 7, performance is confirmed by quantitative metrics measuring recall and precision, and individual components of the training process show increased performance in an ablation study.

8.2 Recommendations for Future Work

Annotated Symmetry Sets The methodology of using rendered segmentation maps to compute symmetry set annotations can be further exploited. In this thesis, we have employed it on symmetries induced by self-occlusion found in the SYMSOL II data set. Nevertheless, we believe the same methodology can be applied when simulating environment-induced occlusions. Moreover, the objects in SYMSOL II appear in a simplistic setting, but the same methodology can be applied to objects in much more complicated environments, like the synthetic images used in the BOP challenge [6]. Finally, future research could seek to improve performance of the method, making it feasible to apply for objects with larger sets of candidate symmetries.

Multiple-Hypothesis Prediction The potential of the EWTA training strategy to increase precision in multiple-hypothesis prediction models has been demonstrated. However, a more robust procedure for decreasing the parameter k is required for its general application. Furthermore, our findings regarding the behaviour of the Von Mises loss function show its potential for avoiding stray hypotheses. Nevertheless, the vanishing gradient in the flat regions of the loss curve can make hypotheses immovable. A scheme in which the parameter $\mathcal{K}$ is evolved during training, is a promising direction for addressing this problem. Furthermore, deeper analyses of the loss function itself, and possible other loss functions with the same properties, would help in understanding their effects in pose estimation.

Binary Classification Though our binary classification model shows impressive performance on the simplistic SYMSOL data set, further research is required into its robustness when objects appear in more complex synthetic environments, and ultimately, in real images. Also, the proposed training strategy only works when symmetry sets have zero volume. More analysis is required to define a category of objects that can satisfy this requirement when considering occlusion-induced symmetries. Alternatively, an extension to the training process to work on symmetry sets with volume would make the method generally applicable. Finally, future research could apply the method to object tracking or extend to a multi-view framework. In both cases, information from multiple view points can be combined by intersecting the symmetry sets predicted from different images.

Bibliography

[1] X. Chen, K. Kundu, Z. Zhang, H. Ma, S. Fidler, and R. Urtasun, "Monocular 3D object detection for autonomous driving," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2147–2156, June 2016.

[2] P. A. Lasota, G. F. Rossano, and J. A. Shah, "Toward safe close-proximity human-robot interaction with standard industrial robots," in *2014 IEEE International Conference on Automation Science and Engineering (CASE)*, pp. 339–344, 2014.

[3] E. Sucar, K. Wada, and A. Davison, "NodeSLAM: Neural object descriptors for multi-view shape reconstruction," in *2020 International Conference on 3D Vision (3DV)*, pp. 949–958, IEEE, 2020.

[4] A. Saxena, J. Driemeyer, and A. Y. Ng, "Robotic grasping of novel objects using vision," *The International Journal of Robotics Research*, vol. 27, no. 2, pp. 157–173, 2008.

[5] T. Hodan, F. Michel, E. Brachmann, W. Kehl, A. GlentBuch, D. Kraft, B. Drost, J. Vidal, S. Ihrke, X. Zabulis, C. Sahin, F. Manhardt, F. Tombari, T.-K. Kim, J. Matas, and C. Rother, "BOP: Benchmark for 6D object pose estimation," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[6] T. Hodaň, M. Sundermeyer, B. Drost, Y. Labbé, E. Brachmann, F. Michel, C. Rother, and J. Matas, "BOP challenge 2020 on 6D object localization," in *Computer Vision – ECCV 2020 Workshops* (A. Bartoli and A. Fusiello, eds.), (Cham), pp. 577–594, Springer International Publishing, 2020.

[7] G. Pitteri, M. Ramamonjisoa, S. Ilic, and V. Lepetit, "On object symmetries and 6D pose estimation from images," in *2019 International Conference on 3D Vision (3DV)*, pp. 614–622, IEEE, 2019.

[8] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: Making RGB-based 3D detection and 6D pose estimation great again," in *Proceedings of the IEEE international conference on computer vision*, pp. 1521–1529, 2017.

[9] M. Rad and V. Lepetit, "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*, pp. 3828–3836, 2017.

[10] Y. Xiang, T. Schmidt, V. Narayanan, and D. Fox, "PoseCNN: A convolutional neural network for 6D object pose estimation in cluttered scenes," *Robotics: Science and Systems (RSS)*, 2018.

[11] I. Gilitschenski, R. Sahoo, W. Schwarting, A. Amini, S. Karaman, and D. Rus, "Deep orientation uncertainty learning based on a bingham loss," in *International Conference on Learning Representations*, 2019.

[12] V. Peretroukhin, M. Giamou, W. N. Greene, D. Rosen, J. Kelly, and N. Roy, "A Smooth Representation of Belief over SO(3) for Deep Rotation Learning with Uncertainty," in *Proceedings of Robotics: Science and Systems*, (Corvalis, Oregon, USA), July 2020.

[13] H. Deng, M. Bui, N. Navab, L. Guibas, S. Ilic, and T. Birdal, "Deep bingham networks: Dealing with uncertainty and ambiguity in pose estimation," *International Journal of Computer Vision*, May 2022.

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

[15] D. Mohlin, J. Sullivan, and G. Bianchi, "Probabilistic orientation estimation with matrix fisher distributions," in *Advances in Neural Information Processing Systems* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), vol. 33, pp. 4884–4893, Curran Associates, Inc., 2020.

[16] B. Okorn, M. Xu, M. Hebert, and D. Held, "Learning orientation distributions for object pose estimation," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 10580–10587, 2020.

[17] X. Deng, A. Mousavian, Y. Xiang, F. Xia, T. Bretl, and D. Fox, "PoseRBPF: A Rao-Blackwellized particle filter for 6D object pose tracking," in *Robotics: Science and Systems (RSS)*, 2019.

[18] K. A. Murphy, C. Esteves, V. Jampani, S. Ramalingam, and A. Makadia, "Implicit-pdf: Non-parametric representation of probability distributions on the rotation manifold," in *Proceedings of the 38th International Conference on Machine Learning*, pp. 7882–7893, 2021.

[19] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger, "Occupancy networks: Learning 3D reconstruction in function space," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4460–4470, 2019.

[20] F. Manhardt, D. M. Arroyo, C. Rupprecht, B. Busam, T. Birdal, N. Navab, and F. Tombari, "Explaining the ambiguity of object detection and 6D pose from visual data," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6841–6850, 2019.

[21] J. Solà, J. Deray, and D. Atchuthan, "A micro lie theory for state estimation in robotics," *arXiv preprint arXiv:1812.01537*, 2018.

[22] R. Szeliski, *Computer Vision - Algorithms and Applications, Second Edition*. Texts in Computer Science, Springer, 2022.

[23] A. Yershova, S. Jain, S. M. LaValle, and J. C. Mitchell, "Generating uniform incremental grids on so(3) using the hopf fibration," *The International Journal of Robotics Research*, vol. 29, no. 7, pp. 801–812, 2010. PMID: 20607113.

[24] J. Diebel, "Representing attitude: Euler angles, unit quaternions, and rotation vectors," technical report, Stanford University, 2006.

[25] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[26] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3D pose estimation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3109–3118, 2015.

[27] K. Park, T. Patten, and M. Vincze, "Pix2Pose: Pixel-wise coordinate regression of objects for 6D pose estimation," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 7668–7677, 2019.

[28] R. Bregier, F. Devernay, L. Leyrit, and J. L. Crowley, "Symmetry aware evaluation of 3D object detection and pose estimation in scenes of many parts in bulk," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[29] L. Beyer, A. Hermans, and B. Leibe, "Biternion nets: Continuous head pose regression from discrete training labels," in *German Conference on Pattern Recognition*, pp. 157–168, Springer, 2015.

[30] M. Sundermeyer, Z.-C. Marton, M. Durner, M. Brucker, and R. Triebel, "Implicit 3D orientation learning for 6D object detection from RGB images," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 699–715, 2018.

[31] T. Hodan, D. Barath, and J. Matas, "EPOS: Estimating 6D pose of objects with symmetries," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11703–11712, June 2020.

[32] J. Richter-Klug and U. Frese, "Handling object symmetries in cnn-based pose estimation," in *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 13850–13856, 2021.

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

[34] C. Rupprecht, I. Laina, R. DiPietro, M. Baust, F. Tombari, N. Navab, and G. D. Hager, "Learning in an uncertain world: Representing ambiguity through multiple hypotheses," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 3591–3600, 2017.

[35] A. Guzman-Rivera, D. Batra, and P. Kohli, "Multiple choice learning: Learning to produce multiple structured outputs," *Advances in neural information processing systems*, vol. 25, 2012.

[36] A. Guzman-Rivera, P. Kohli, B. Glocker, J. Shotton, T. Sharp, A. Fitzgibbon, and S. Izadi, "Multi-output learning for camera relocalization," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1114–1121, 2014.

[37] D. Dey, V. Ramakrishna, M. Hebert, and J. Andrew Bagnell, "Predicting multiple structured visual interpretations," in *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2947–2955, 2015.

[38] K. Lee, C. Hwang, K. Park, and J. Shin, "Confident multiple choice learning," in *International Conference on Machine Learning*, pp. 2014–2023, PMLR, 2017.

[39] K. Tian, Y. Xu, S. Zhou, and J. Guan, "Versatile multiple choice learning and its application to vision computing," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 6349–6357, 2019.

[40] O. Makansi, E. Ilg, O. Cicek, and T. Brox, "Overcoming limitations of mixture density networks: A sampling and fitting framework for multimodal future prediction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 7144–7153, 2019.

[41] Y. Zhou, C. Barnes, J. Lu, J. Yang, and H. Li, "On the continuity of rotation representations in neural networks," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5745–5753, 2019.

[42] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[43] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pp. 248–255, 2009.

[44] T. Hodan, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis, "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)*, pp. 880–888, IEEE, 2017.

[45] Y. Xiang, R. Mottaghi, and S. Savarese, "Beyond PASCAL: A benchmark for 3D object detection in the wild," in *IEEE Winter Conference on Applications of Computer Vision*, pp. 75–82, 2014.

[46] S. Liao, E. Gavves, and C. G. M. Snoek, "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 (CVPR)*, June 2019.

[47] J. Arvo, "Iii.4 - fast random rotation matrices," in *Graphics Gems III (IBM Version)* (D. KIRK, ed.), pp. 117–120, San Francisco: Morgan Kaufmann, 1992.

[48] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2015.

[49] Z. Chen and H. Zhang, "Learning implicit fields for generative shape modeling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 5939–5948, 2019.

[50] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, "PointNet: Deep learning on point sets for 3D classification and segmentation," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 652–660, July 2017.

[51] N. Rahaman, A. Baratin, D. Arpit, F. Draxler, M. Lin, F. Hamprecht, Y. Bengio, and A. Courville, "On the spectral bias of neural networks," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 5301–5310, PMLR, 09–15 Jun 2019.

[52] B. Mildenhall, P. P. Srinivasan, M. Tancik, J. T. Barron, R. Ramamoorthi, and R. Ng, "Nerf: Representing scenes as neural radiance fields for view synthesis," in *Computer Vision – ECCV 2020* (A. Vedaldi, H. Bischof, T. Brox, and J.-M. Frahm, eds.), (Cham), pp. 405–421, Springer International Publishing, 2020.

[53] C. J. van Rijsbergen, *Information retrieval*. Butterworth, 1979.

Appendix A

Embedded Loss Function Analysis

To re-iterate, the definitions of the embedded loss functions are as follows:

$$\mathcal{L}_{COS}(\alpha) = 1 - \cos(\alpha) \tag{A.1}$$

$$\mathcal{L}_{VM}(\alpha) = 1 - e^{\mathcal{K}(\cos(\alpha)-1)} \ , \tag{A.2}$$

where α is the smallest angular difference between a predicted rotation R_p and a ground truth rotation R_{GT}. Moreover, $\mathcal{K}$ is a hyperparameter of the Von Mises distribution, which is analogous to the inverse of the standard deviation of the Gaussian distribution. Note that the smallest angular difference is computed as

$$\cos(\alpha) = \frac{\big(\mathrm{Tr}(R_p R_{gt}^\intercal) - 1\big)}{2} \ , \tag{A.3}$$

thus defining the actual loss functions as

$$\mathcal{L}_{COS}(R_{GT}, R_p) = 1 - \frac{\big(\mathrm{Tr}(R_p R_{gt}^\intercal) - 1\big)}{2} \tag{A.4}$$

$$\mathcal{L}_{VM}(R_{GT}, R_p) = 1 - e^{\mathcal{K}\left(\frac{(\mathrm{Tr}(R_p R_{gt}^\intercal)-1)}{2}-1\right)} \ . \tag{A.5}$$

A.1 Comparing Shapes

Since the behaviour of these loss function in 3D is rather difficult to visualize and analyze, we instead start by looking at the $2D$ case. This reduces the complexity, since poses in 2D can simply be described by a single angle, and their smallest angular difference is simply their difference in angle. Given two rotations described by the angles α and β, the same loss functions can be described in 2D as

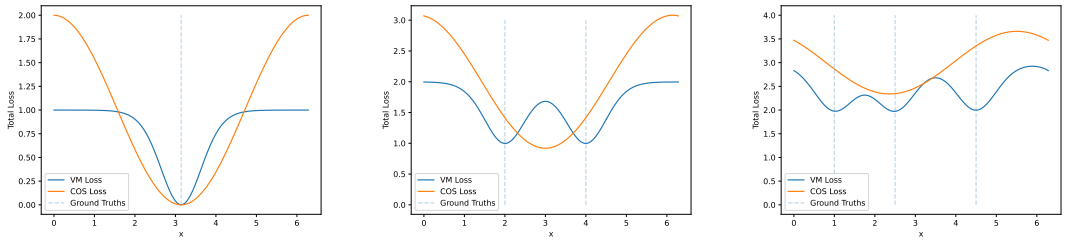
$$\mathcal{L}_{COS}(\alpha, \beta) = 1 - \cos(\alpha - \beta) \tag{A.6}$$

$$\mathcal{L}_{VM}(\alpha, \beta) = 1 - e^{\mathcal{K}(\cos(\alpha-\beta)-1)} \ . \tag{A.7}$$

Say we have a simple 2D example setup, where a network is trained to predict all symmetric rotations for a single image. As was mentioned before, at the start of the EWTA training process, the loss is computed between each hypothesis and each ground truth symmetry. Consequently, the total loss that a single hypothesis x contributes given a set of ground truth symmetries Y is given by the sum

$$\mathcal{L}_\Sigma(x) = \sum_{y \in Y} \mathcal{L}(x, y)$$

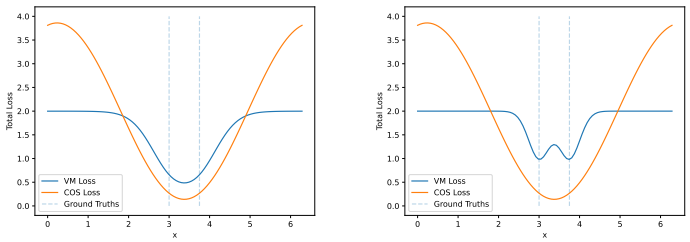
, where $\mathcal{L}$ is the embedded loss function. Figure A.1 plots $\mathcal{L}_\Sigma(x)$ for three different example sets Y using both $\mathcal{L}_{COS}$ and $\mathcal{L}_{VM}$. Here, $\mathcal{L}_{VM}$ uses $\mathcal{K} = 4$, which is the value used in the main text. Interestingly, the summed loss using $\mathcal{L}_{VM}$ has minima very close to the ground truth rotations, whereas using $\mathcal{L}_{COS}$, the loss is minimized in some conditional average.



(a) (b) (c)

Figure A.1: Plots of $\mathcal{L}_\Sigma(x)$ using both $\mathcal{L}_{COS}$ and $\mathcal{L}_{VM}$ for three example sets Y: (a) $\{\pi\}$, (b) $\{2, 4\}$, and (c) $\{1, 2.5, 4.5\}$. In $\mathcal{L}_{VM}$ uses $\mathcal{K} = 4$.

When the ground truth samples are close enough, however, the distinct minima of the Von Mises loss do collapse into a single average (Figure A.2a). By increasing $\mathcal{K}$ and, therefore, increasing the steepness of the Von Mises, the minima are again separated (Figure A.2b). Note that the loss plot using the Von Mises loss become very flat in the regions far away from ground truth samples, which becomes worse for increased values of $\mathcal{K}$. This can be problematic in deep learning, as predictions in these regions will receive small gradients, thus preventing them from moving.



(a) $k = 4$ (b) $k = 15$

Figure A.2: Plots of $\mathcal{L}_\Sigma(x)$ using both $\mathcal{L}_{COS}$ and $\mathcal{L}_{VM}$ for $Y = \{3, 3.75\}$, with varying k in the Von Mises loss.

A.2 Finding the Minima

The exact location of the minimum of $\mathcal{L}_\Sigma$ can be found analytically by setting the derivative to zero, and solving for x.

A.2.1 Cyclic Cosine Loss

When using $\mathcal{L}_{COS}$, this is relatively easy. We first find the derivative of $\mathcal{L}_\Sigma$ with respect to x as

$$\begin{aligned}
\frac{d\mathcal{L}_\Sigma(x)}{dx} &= \sum_{y\in Y} \frac{d\mathcal{L}_{COS}(x,y)}{dx} && \text{(A.8)}\\
&= \sum_{y\in Y} \sin(x-y) && \text{(A.9)}\\
&= \sum_{y\in Y} \sin(x)\cos(y) - \cos(x)\sin(y) && \text{(A.10)}\\
&= \sum_{y\in Y} \sin\cos(y) - \sum_{y\in Y} \cos(x)\sin(y) && \text{(A.11)}\\
&= \sin(x)\sum_{y\in Y} \cos(y) - \cos(x)\sum_{y\in Y} \sin(y)\ . && \text{(A.12)}
\end{aligned}$$

Now, setting it equal to 0 to find the location of extrema

$$\begin{aligned}
&\frac{d\mathcal{L}_\Sigma(x)}{dx} = 0 && \text{(A.13)}\\
\iff &\sin(x)\sum_{y\in Y} \cos(y) - \cos(x)\sum_{y\in Y} \sin(y) = 0 && \text{(A.14)}\\
\iff &\frac{\sin(x)}{\cos(x)} = \frac{\sum_{y\in Y}\sin(y)}{\sum_{y\in Y}\cos(y)} && \text{(A.15)}\\
\iff &x = \tan^{-1}\left(\frac{\sum_{y\in Y}\sin(y)}{\sum_{y\in Y}\cos(y)}\right) + \pi n\ . && \text{(A.16)}
\end{aligned}$$

This gives exactly one maximum and one minimum within the domain of $[0, 2\pi]$.

A.2.2 Von Mises Loss

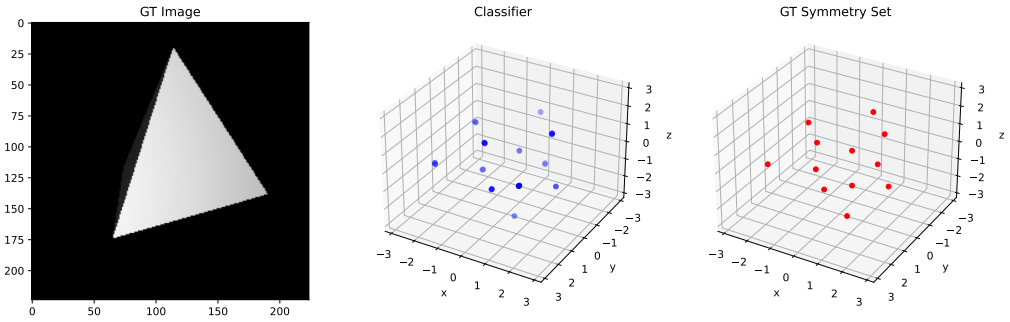
Doing the same with with $\mathcal{L}_{VM}$ is rather more difficult. Finding the derivative is easy enough.

$$\begin{aligned}
\frac{d\mathcal{L}_\Sigma(x)}{dx} &= \sum_{y\in Y} \frac{d\mathcal{L}_{VM}(x,y)}{dx}\\
&= \sum_{y\in Y} \mathcal{K}e^{\mathcal{K}(\cos(x-y)-1)}\sin(x-y)
\end{aligned}$$

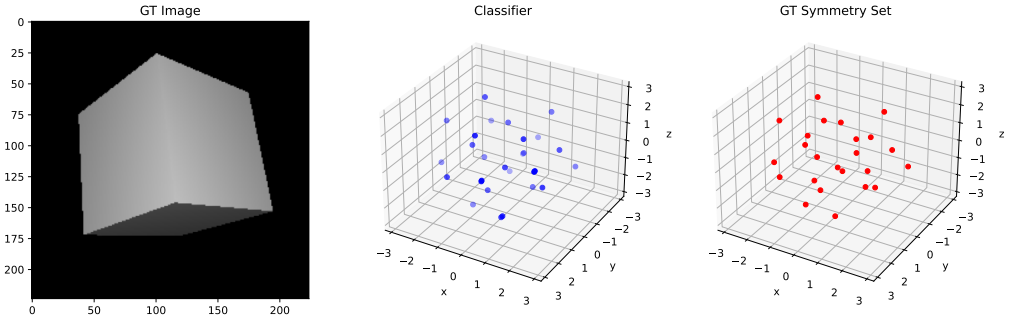
But setting it equal to 0 and solving for x is much more complicated. Nevertheless, the properties of the Von Mises loss shown Figure A.1 do illustrate how multiple minima can exist. These minima lie a small distance away from the individual elements in Y, and this distance was found to shrink for increased values of $\mathcal{K}$. An intuitive explanation as to why these multiple minima exist, concerns the shape the loss function. When summing multiple losses together, the minima of individual losses can fall inside the flat regions of others. When this happens, the graphs are essentially stacked, and multiple minima occur in the summed loss. These minima are, however, slightly offset from their original positions, as the flat regions are not perfectly flat. Increasing $\mathcal{K}$ further flattens the loss curves, therefore decreasing the size of this offset.

Appendix B

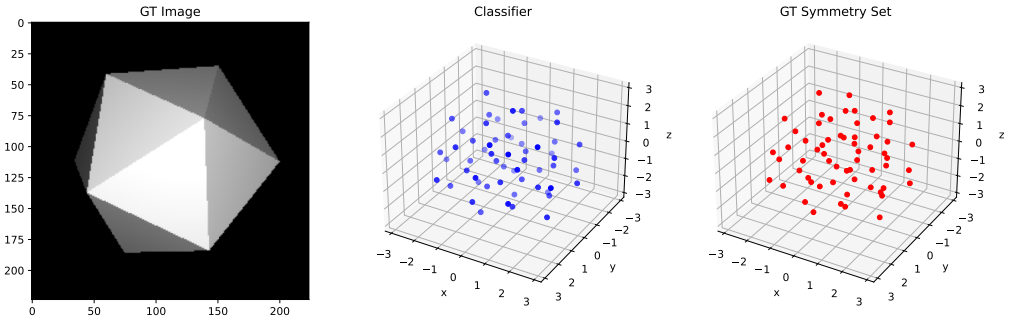
Plots for Binary Classifiers



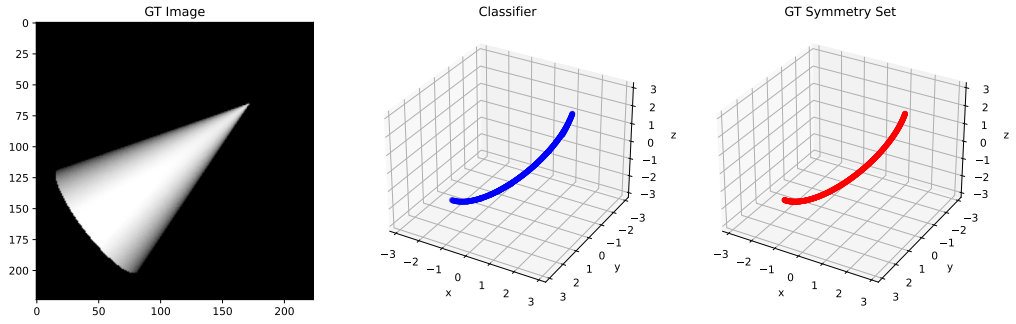
(a) Tetrahedron



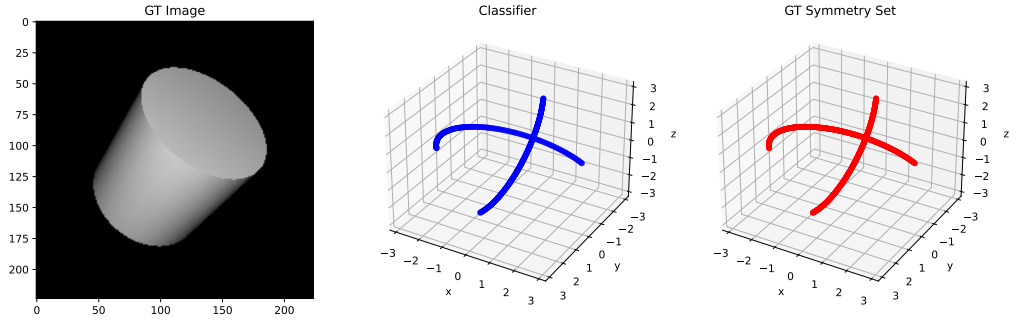
(b) Cube



(c) Icosahedron

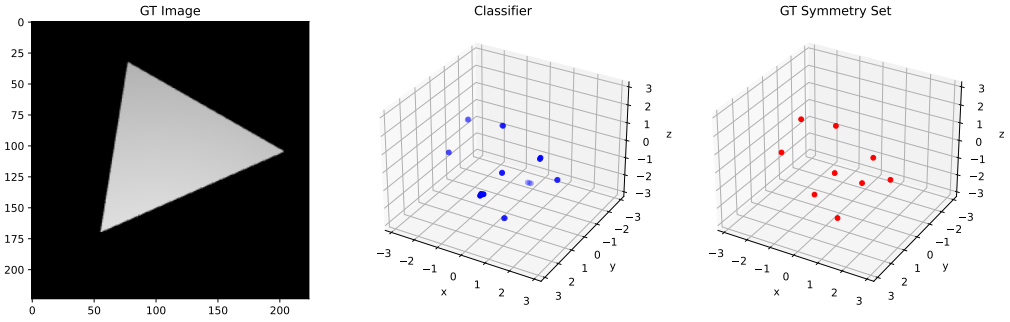


(d) Cone

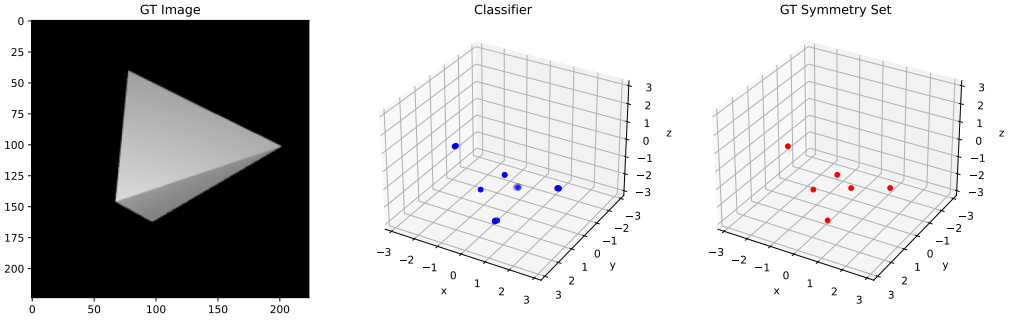


(e) Cylinder

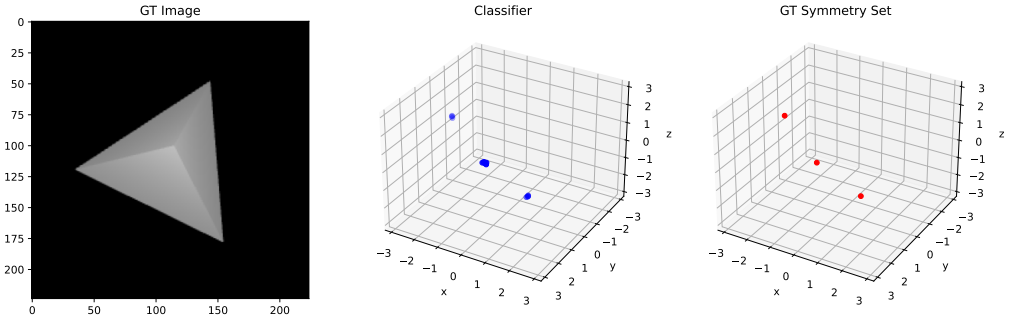
Figure B.1: π-ball plots showing outputs for the baseline model from Section 7.4.1 on images from SYMSOL I. All rotations in $S(I) \cup \mathcal{G}$ are input to the classifier for each image, only positive predictions are shown in the middle column.



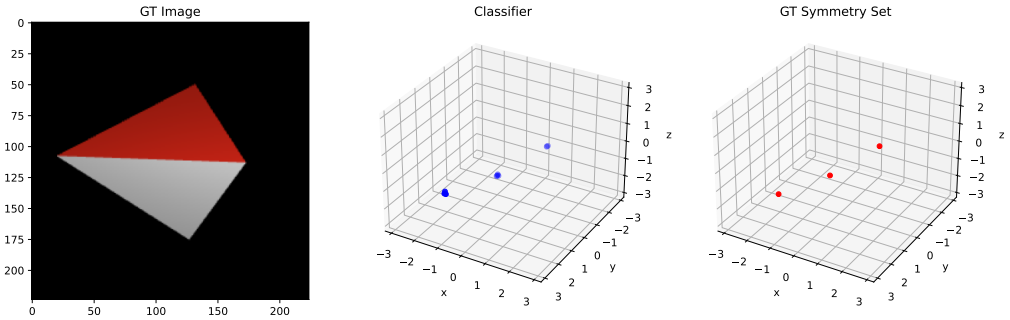
(a) One gray face is visible.



(b) Two gray face are visible.

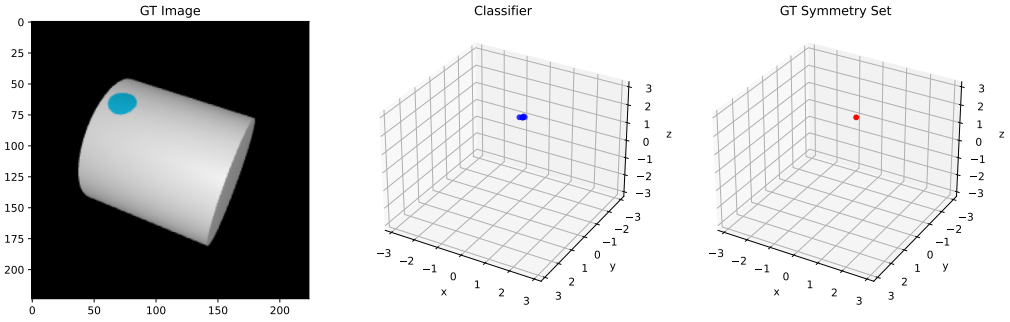


(c) Three gray faces are visible.

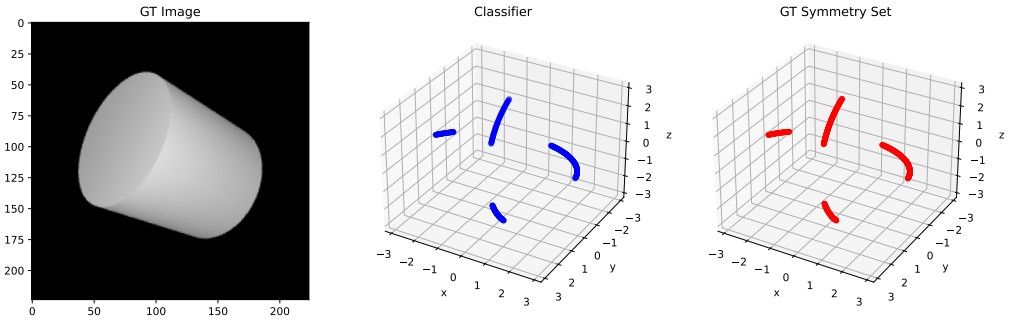


(d) The red face is visible.

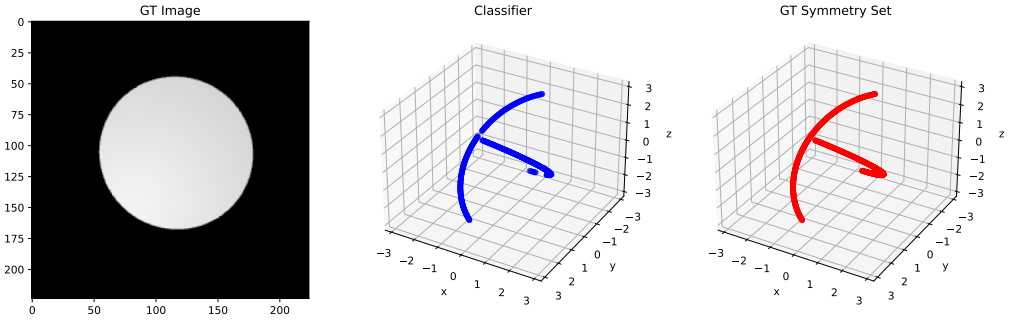
Figure B.2: π-ball plots showing outputs for the baseline model from Section 7.4.1 on images of the marked tetrahedron from SYMSOL II. The marked tetrahedron exhibits a different number of rotational symmetries depending on which faces are visible. All rotations in $S(I) \cup \mathcal{G}$ are input to the classifier for each image, only positive predictions are shown in the middle column.



(a) The mark is visible.



(b) The mark is not visible.



(c) A view from the top or bottom.

Figure B.3: π-ball plots showing outputs for the baseline model from Section 7.4.1 on images of the marked cylinder from SYMSOL II. The marked cylinder exhibits different symmetries based on whether the marking is visible. All rotations in $S(I) \cup \mathcal{G}$ are input to the classifier for each image, only positive predictions are shown in the middle column.