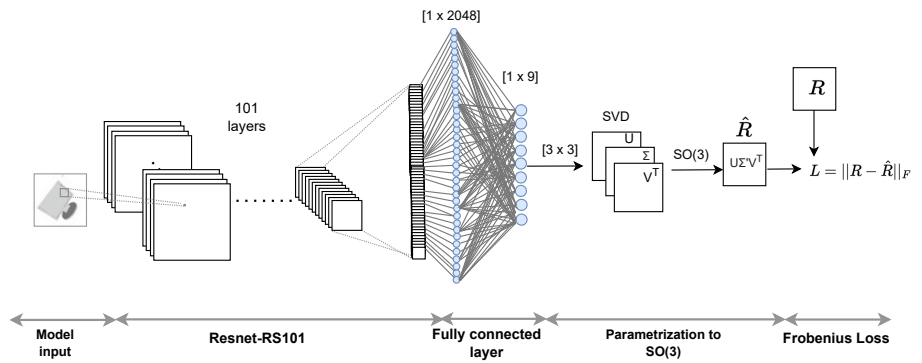


Henrik Grüner

Rotation Representation Methods for Pose Estimation with Deep Learning

Master's thesis in Engineering and ICT
 January 2022



Henrik Grüner

Rotation Representation Methods for Pose Estimation with Deep Learning

Master's thesis in Engineering and ICT
January 2022

Norwegian University of Science and Technology
Faculty of Engineering
Department of Mechanical and Industrial Engineering

Preface

The report is a culmination of five years of studying Engineering and ICT and is the result of my Master's thesis in the spring of 2022. The report is written for the Department of Mechanical and Industrial Engineering at the Norwegian University of Science and Technology in robotics and automation.

Acknowledgements

I want to use this opportunity to thank my supervisor, Prof. Olav Egeland, for taking the role of my supervisor. He has given me theoretical guidance while writing the thesis and the flexibility needed to finish an M.Sc. in Financial Economics. Further, I would like to thank Ola Alstad for vital help with different software for 3D rendering and an introduction to various problems in pose estimation.

Summary

The ability to regress 3D rotation matrices is an old problem in the field of computer vision. With the rise of deep learning methods, researchers have attempted to leverage neural networks for regression on rotation matrices. The task is difficult due to topological differences between the rotations and output of the models. Neural networks usually output data in the Euclidean space, whereas rotations in 3D are represented by the special orthogonal group $SO(3)$, which is not topological homeomorphic to any subset of real 4D Euclidean space. This mismatch calls for a mapping function between the model and the estimated rotation output.

The traditional methods for representing rotations, such as Euler angles and quaternions, have dimensions 3 and 4, respectively, and are hence discontinuous in the real space. Other representation functions such as Gram-Schmidt orthogonalization and symmetric orthogonalization with singular value decomposition (SVD) have been proposed as a solution. This thesis gives an overview of the state of rotation matrix regression, the desirable properties for a mapping function from real Euclidean space to non-Euclidean manifolds, and shows that the SVD orthogonalization is the mapping with the best performance. The theoretical arguments are strengthened by experiments using state-of-the-art deep learning methods. Both a comparison experiment is conducted, in addition to experiments showcasing the potential of symmetric orthogonalization in pose estimation problems.

Contents

Preface	i
Summary	iii
1. Introduction and Motivation	3
1.1. Related work	3
1.2. Objectives	4
2. Prerequisites	7
2.1. The Adjoint Operator	7
2.2. Preimage	7
2.3. Surjectivity	7
2.4. Injection	7
2.5. Bijection	8
2.6. Homeomorphism	8
2.7. Image Representation	8
3. Linear Algebra	11
3.1. Definitions	11
3.1.1. Symmetric Matrix	11
3.1.2. Skew Symmetric Matrix	11
3.1.3. The Hadamard Product	12
3.1.4. The Trace of a Matrix	12
3.1.5. Derivatives of the Trace Operator	13
3.1.6. The Frobenious Norm	13
3.1.7. Frobenious Inner Product	13
3.1.8. Orthogonal set	15
3.1.9. Orthogonal group	15
3.2. Eigendecomposition	15
3.3. Singular Value Decomposition	16
3.4. Jacobian matrix	18
3.5. Matrix differentials	18
3.5.1. Differentials	18

4. Artificial Neural Networks	21
4.1. The Artificial Neuron	21
4.2. Fully Connected Feed Forward Network	23
4.3. Convolutional Neural Network	24
4.3.1. Convolutional Operator	25
4.3.2. The Convolutional Layer	26
4.4. Learning the Parameters - intuition	27
4.4.1. Convergence Under Empirical Risk Minimization	29
4.5. Backpropagation	29
4.6. Matrix Backpropagation	30
4.6.1. Partial derivatives	30
4.7. Backpropagation for a SVD layer	31
5. Transformations and coordinate frames	35
5.1. Rotation Matrix	35
5.1.1. Properties	36
5.1.2. Representing an Orientation	37
5.1.3. Changing the Frame	38
5.2. Comparing Rotation Matrices	39
5.2.1. Angle error	39
5.2.2. Geodesic Loss	39
5.3. Camera Model	39
5.3.1. The Pinhole Camera Model	39
5.3.2. Camera Rotation and Translation	41
5.4. Pose Estimation	42
6. Mappings from Real Euclidean Space to Special Orthogonal Group	45
6.1. Desired Properties	45
6.2. Differentiable Mappings - Examples	47
6.2.1. Euler Angles	47
6.3. Quaternions	48
6.3.1. Properties	48
6.3.2. Rotation Through Quaternions	49
6.4. Gram-Schmidt Orthogonalization	50
6.4.1. 5D and 6D representations of rotations	50
6.5. Symmetric Orthogonalization via SVD	51
6.5.1. Wahba's Problem	52
6.5.2. Procrustes Problem	53
6.5.3. Properties of Symmetric Srthogonalization	55
6.5.4. Gradients	55
6.5.5. Comparison with Gram-Schmidt	57

6.6. Summary	58
7. Experiments	59
7.1. Datasets	59
7.1.1. ModelNet	59
7.1.2. UPNA	61
7.2. Comparison test	61
7.3. 3D Head Pose Estimation from 2D images of humans	63
7.3.1. Setup & Pre-Processing	63
7.4. 3D Pose Estimation from 2D Images	64
7.4.1. Network Architecture	64
7.5. 3D Iterative Pose alignment	65
7.5.1. Network Architecture	66
8. Results and discussion	69
8.1. Comparison Test for Mapping Functions	69
8.2. UPNA Head Pose	74
8.3. 3D object pose estimation from 2D images	75
8.3.1. Symmetry in the samples	75
8.4. Iterative 3D Pose Refinement	78
9. Conclusion	81
A. Supplementary material	89
A.1. Convolutions	89
B. 3D Pose Estimation - Supplementary Materials	93
B.1. Distribution of Angle Error	93

List of Figures

2.1. Gray Scale Image Representation	9
2.2. RBG-Image	9
4.1. Artificial Neuron	22
4.2. Rectified Linear Unit and the Logistic Sigmoid	23
4.3. Fully connected neural network	25
4.4. Convolved 2D image	26
4.5. Gradient descent	28
5.1. Orientation with two Rotation matrices	37
5.2. Coordination frame rotated	38
5.3. The Pinhole camera model	40
5.4. Euclidean transformation	41
5.5. Illustration of Pose Estimation Problem	43
6.1. Wahba's Problem	53
7.1. ModelNet10 Classes visualized	60
7.2. Point Cloud Examples from ModelNet10	60
7.3. UPNA Head Pose Samples	61
7.4. Rotation axes of a Human Head	61
7.5. Network Architecture:	62
7.6. One sample from the UPNA data set cropped to 224×224	63
7.7. Overview of Model Architecture	65
7.8. 3D Iterative Pose Estimation: Network Architecture	67
8.1. The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 45 per axis.	70
8.2. The mean angle error epoch 25-50 max 45 degrees	71
8.3. The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 45 per axis.	71
8.4. The mean angle error between the true rotation matrix restricted to 0-4 degrees from epoch 25-50 with rotations up to 90 per axis.	72

8.5.	The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 180 per axis.	72
8.6.	The mean angle error between the true rotation matrix restricted to 0-4 degrees from epoch 25-50 with rotations up to 180 per axis.	73
8.7.	Training and Test angle error UPNA	74
8.8.	Distribution of angle errors before and after adjusting for symmetry.	76
8.9.	Results 3D pose estimation	77
8.10.	Results from the Iterative Pose Refinement	79
A.1.	Convolution example	90
A.2.	Padded convolution	91
A.3.	Padded convolution with stride = 2	92
B.1.	Overlay between ground truth pose and rendered image using the estimated rotation matrix.	94
B.2.	The distribution of angle error for sofa class	95
B.3.	The distribution of angle error for bed class	95
B.4.	The distribution of angle error for bathtub class	96
B.5.	The angle errors for the table class	96
B.6.	The distribution of angle errors for the toilet class	97
B.7.	The distribution of angle error for desk class	97
B.8.	The distribution of angle error for monitor class	98
B.9.	The distribution of angle error for night stand class	98
B.10.	The distribution of angle error for dresser class	99
B.11.	The distribution of angle error for chair class	99

List of Tables

6.1. Table showcasing which of the desirable properties each mapping function satisfies.	58
8.1. Mean angle errors for preliminary experiment	69
8.2. 3D pose estimation - all classes	75
8.3. 3D pose estimation Modelnet10 results	77
8.4. Symmetry Adjusted 3D pose estimation	77
8.5. Performance of the Iterative Refinement model	78
B.1. 3D pose estimation - all classes	93
B.2. The angle errors for the sofa class	95
B.3. The angle errors for the bed class	95
B.4. The distribution of angle errors for the bathtub class	96
B.5. The distribution of angle errors for the table class	96
B.6. The distribution of angle errors for the toilet class	97
B.7. The angle errors for the desk class	97
B.8. The angle errors for the monitor class	98
B.9. The angle errors for the night stand class	98
B.10. The angle errors for the dresser class	99
B.11. The angle errors for the chair class	99

Acronyms

$O(n)$ Orthogonal group. [15](#)

$SO(n)$ Special Orthogonal group. [15](#)

ANN Artificial Neural Network. [21](#)

CNN Convolutional Neural Network. [24](#)

MBP Matrix Back-propagation. [30](#)

SVD Singular Value Decomposition. [16](#)

Notation

Upper case bold Latin letters are used for matrices, whilst regular upper case letters are reserved for sets. Lowercase case Latin letters are used primarily for vectors but also for scalars. During ambiguity, the dimension will be denoted.

Chapter 1.

Introduction and Motivation

Rotations are a ubiquitous part of computer vision problems. For every rigid body in graphics, simulations, or problems involving rigid objects such as autonomous driving, rotations describe orientation and motion. The rotation representation is a fundamental part of how robots can orient themselves in 3D space, both directly and indirectly, through their cameras. The number of applications and technology based on rotations is enormous. Hence the ability to estimate its own and other objects poses from 2D images is an important ability for a lot of robots and applications. 3D pose estimation ranges from autonomous orientation [34, 46] augmented and virtual reality [1, 31], and 3D reconstruction applications and visual odometry [12, 7, 50]. The list of fields relying on rotations and pose estimation is endless, but they all have one item in common: The quality and function of the technology are heavily dependent on the performance of the underlying orientation estimation process.

1.1. Related work

Optimization on Riemann manifolds and, more specifically, the $SO(3)$ group is a well-studied problem but has yet to find a general solution. The difficulty stems from $SO(3)$ is not topologically homeomorphic to any subset of 4D Euclidean space. This affects all the traditional rotation representations, including Euler angles, axis-angle, and unit quaternions. This mismatch can be circumvented by using a differentiable function from a higher dimension, mapping the inputs onto the $SO(3)$ manifold.

One of the first approaches with deep learning methods was to estimate Euler angles with a classification model rather than a direct regression problem. The idea was to discretize the angles into bins and apply softmax on the outputs to normalize the angles to a probability distribution [45, 47]. Discretizing the out-

puts into bins increases the dimensionality and expressivity of the problem, hence making the methods sub-optimal compared to regression methods. The classification approach was extended to use the expectations of the discrete distributions as a continuous angle for regression [21]. In [27] the switch is made to quaternions rather than Euler angles. The authors note the seeming paradox of using a discrete classification method on a continuous problem; nevertheless, they show that soft-maxing discrete outputs lead to more stable training than direct regression. To stabilize the training during unconstrained continuous regression, the authors introduce a “spherical exponent” mapping function to improve the stability during the training.

[42] demonstrates that quaternions are not suitable for learning problems due to almost equal rotations being far away from each other as quaternions. However, no alternative representations were proposed except direct regression, which may not produce valid rotation matrices. In [52] the authors note that there was an abnormally high presence of errors between 90 and 180 degrees, even for non-rotational symmetric objects. In [54], the authors argue that these errors are caused by the discontinuity from the rotation representations of Euler angles and quaternions. Zhou et al. proposes the Gram-Schmidt orthogonalization method as a mapping from real Euclidean space to the special orthogonal group and shows the continuous properties of the method. This analysis is taken one step further in [24], which explores the viability of singular value decomposition orthogonalization (also referred to as Procrustes) as a mapping to $SO(3)$. It is proved that the mapping is continuous, and the authors argue that the natural choice for projecting onto $SO(3)$ and quickly achieved state-of-the-art results for 3D pose estimation tasks. [6] argues that the continuity criterion for a suitable mapping is too loose of a criterion. The author highlights a set of desirable properties for mapping onto $SO(3)$, focusing on pre-image connectivity, and demonstrates that SVD orthogonalization outperforms the other mapping function.

1.2. Objectives

The objective of this thesis is two-fold:

1. Discuss and research the theoretical arguments for the different mappings from real Euclidean space \mathbb{R}^n to $SO(3)$
2. Run experiments to verify the findings of the theoretical section using state-of-the-art machine learning architecture.

The thesis aims to give a reader unacquainted with rotations and machine learning the essential tools to understand the problem and the solutions. Hence a chapter about linear algebra in addition to one on the foundations for rotations in robotics.

Further, the results and the theoretical arguments require a high understanding of machine learning. For this motivation, a chapter explaining the foundations of neural networks is included. This chapter may be skipped if the reader has high expertise in the area.

Chapter 2.

Prerequisites

2.1. The Adjoint Operator

For any linear operator \mathbf{A} , the adjoint \mathbf{A}^* is defined as:

$$\langle v, \mathbf{A}u \rangle = \langle \mathbf{A}^*v, u \rangle \quad (2.1)$$

2.2. Preimage

Let $f : X \rightarrow Y$ be a function and $A \in Y$. Then the preimage of A under f is denoted as

$$f^{-1}(A) = \{x \in X : f(x) \in A\} \quad (2.2)$$

2.3. Surjectivity

A function $f : A \rightarrow B$ is surjective on B if $f(A) = B$, i.e., for every element in $b \in B$, there is an element $a \in A$ such that

$$f(a) = b \quad (2.3)$$

Surjectivity can be seen as it is possible to reach every element in B with the function f and A .

2.4. Injection

A function f is said to be injective, or one-to-one, if it maps distinct elements to distinct elements. Every element of the functions codomain is the image of at

most one element in the domain of the function.

2.5. Bijection

A function is a bijection if it is surjective and injective.

2.6. Homeomorphism

Homeomorphism refers to a correspondence between two geometrical objects or surfaces. A function $f : X \rightarrow Y$ between two topological spaces are said to be a homeomorphism if it satisfies the following properties:

1. f is a bijection
2. f is continuous
3. The inverse f^{-1} is continuous.

If such a function f exists, then the spaces X and Y are a homeomorphism. This can be seen as a equivalence relation, and intuitively, the spaces can be seen as the same.

2.7. Image Representation

Every image displayed on a computer consists of pixels which can be interpreted as the intensity value. Consider a black and white image: The image is represented as a matrix consisting of pixels in the interval $[0, 255]$ representing the amount of white in the image. Figure 2.1 illustrates how an image of a pair of pants from the grayscale Fashion MNIST [53] data set is represented.

For images with coloring, the most popular representation is RGB. RGB-images have three matrices representing the intensity of the colors red, green, and blue (RGB) for every location. The mixture of these three colors yields $256^3 = 16777216$ different colors. Each of these matrices is referred to as a channel. Even though RGB images are technically three-dimensional due to the color hues, they are seen as two-dimensional.

0	0	0	0	0	0	0	0	0	0	0	192	188	181	189	157	165	188	176	179	181	0	0	0	0	0	0	0	0	0	
0	0	0	0	0	0	0	0	0	0	0	14	235	213	213	223	225	217	214	204	211	189	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	22	234	217	211	217	200	203	217	203	179	170	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	105	252	205	209	219	193	195	212	207	194	204	51	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	204	228	184	206	215	221	212	209	199	190	227	105	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	244	192	193	212	217	210	216	216	194	175	225	131	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	25	247	170	192	210	211	239	231	219	199	179	217	124	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	49	236	170	193	201	220	187	231	230	206	181	213	116	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	87	232	159	193	200	245	0	207	245	200	176	210	109	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	127	221	170	189	216	225	0	151	254	198	183	209	103	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	163	206	176	185	242	146	0	53	255	201	183	205	100	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	177	197	175	189	247	34	0	0	253	203	180	199	97	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	172	186	174	199	223	0	0	0	234	204	162	199	106	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	154	177	174	211	151	0	0	0	210	202	141	180	108	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	141	171	175	246	70	0	2	0	177	204	121	145	136	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	135	179	191	248	28	0	5	0	134	210	107	142	119	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	90	189	198	211	4	0	3	0	59	244	118	165	123	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	3	187	195	225	70	0	4	0	20	215	116	167	137	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	182	199	221	163	0	0	0	0	238	147	179	129	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	194	208	213	217	0	0	0	0	246	197	203	138	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	135	225	206	201	0	0	0	0	215	215	221	140	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	27	219	204	217	34	0	0	0	202	215	224	123	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	203	214	223	88	0	0	0	196	219	230	130	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	133	217	217	167	0	0	0	184	221	232	132	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	3	0	59	218	209	195	0	0	0	169	228	214	150	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	1	0	0	191	206	219	47	0	0	155	237	223	133	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	161	238	234	116	0	0	168	246	237	156	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0	38	137	132	53	0	0	81	145	129	62	0	0	0	0	0	0	0	0

Figure 2.1.: $1 \times 28 \times 28$ image of a pair of pants from the Fashion MNIST data set [53]. The value of 255 represents the colour white, whereas the number 0 is black.



Figure 2.2.: a $3 \times 3000 \times 4000$ RGB-image, where each channel corresponds to the amount of red, blue and green in each pixel. Every channel has values ranging from 0-255, depending on the intensity of the pixel. Photograph taken by me

Chapter 3.

Linear Algebra

3.1. Definitions

3.1.1. Symmetric Matrix

A matrix \mathbf{A} is said to be symmetric[2] if and only if

$$\mathbf{A} = \mathbf{A}^T \quad (3.1)$$

A symmetric matrix is necessarily square, and it is only symmetric if and only if

$$a_{ij} = a_{ji}, \quad \forall i, j \quad (3.2)$$

The definition yields that:

$$\mathbf{A} - \mathbf{A}^T = 0 \quad (3.3)$$

3.1.2. Skew Symmetric Matrix

A matrix \mathbf{A} is said to be skew symmetric [2] (also known as anti-symmetric) if and only if:

$$\mathbf{A} = -\mathbf{A}^T \quad (3.4)$$

A skew symmetric matrix is necessarily square, and it is only symmetric if and only if

$$a_{ij} = -a_{ji}, \quad \forall i, j \quad (3.5)$$

All of the main diagonal entries must be zero of a skew-symmetric matrix, as

$$a_{ii} = -a_{ii} \iff a_{ii} = 0 \quad (3.6)$$

By following the definition, the neat property follows:

$$\mathbf{A} + \mathbf{A}^T = 0 \quad (3.7)$$

3.1.3. The Hadamard Product

The Hadamard product (also known as element-wise multiplication) is the product of corresponding entries in two matrices [35]. For two matrices \mathbf{A} and \mathbf{B} of the dimensions $m \times n$, the Hadamard product $\mathbf{A} \circ \mathbf{B}$ is a matrix of the same dimensions as the operands with elements given by:

$$(\mathbf{A} \circ \mathbf{B})_{ij} = \mathbf{A}_{ij} \mathbf{B}_{ij} \quad (3.8)$$

An example where \mathbf{A}, \mathbf{B} with dimensions 2×2 :

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \circ \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} b_{11} & a_{12} b_{12} \\ a_{21} b_{21} & a_{22} b_{22} \end{bmatrix} \quad (3.9)$$

Note that the Hadamard product requires the matrices to be of same size.

3.1.4. The Trace of a Matrix

The trace of a matrix \mathbf{A} is denoted $\text{tr}(\mathbf{A})$ and is equal to the sum of the elements on the diagonal:

$$\text{tr}(\mathbf{A}) = \sum_{i=1}^n a_{ii} = a_{11} + a_{22} + \cdots + a_{nn} \quad (3.10)$$

Some of the the properties of the trace operator is:

$$\text{tr}(\mathbf{A} + \mathbf{B}) = \text{tr}(\mathbf{A}) + \text{tr}(\mathbf{B}) \quad (3.11)$$

$$\text{tr}(c\mathbf{A}) = c \text{tr}(\mathbf{A}) \quad (3.12)$$

$$\text{tr}(\mathbf{A}) = \text{tr}(\mathbf{A}^T) \quad (3.13)$$

$$\text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA}) \quad (3.14)$$

$$\text{tr}(\mathbf{X}^T \mathbf{Y}) = \text{tr}(\mathbf{XY}^T) = \text{tr}(\mathbf{YX}^T) = \text{tr}(\mathbf{Y}^T \mathbf{X}) \quad (3.15)$$

For all square matrices \mathbf{A} and \mathbf{B} , and all scalars c .

Further, let $\mathbf{A} \in \mathbb{R}^{1 \times k}$ and $\mathbf{B} \in \mathbb{R}^{k \times 1}$, then the following is true:

$$\mathbf{AB} = \text{tr}(\mathbf{AB}) = \text{tr}(\mathbf{BA}) \quad (3.16)$$

The trace of a scalar is equal to the scalar (think of the scalar as a 1×1 matrix with one unique diagonal element).

3.1.5. Derivatives of the Trace Operator

Let \mathbf{X} and \mathbf{A} be two square matrices. Then let:

$$f(\mathbf{X}) = \text{tr}(\mathbf{X}\mathbf{A}) \quad (3.17)$$

Which can be written as

$$f(\mathbf{X}) = \text{tr} \left(\sum_{m=1}^n x_{km} a_{ml} \right) = \sum_{k=1}^n \sum_{m=1}^n x_{km} a_{mk} \quad (3.18)$$

The equation w.r.t \mathbf{X} :

$$\frac{df(\mathbf{X})}{d\mathbf{X}} = \left\{ \frac{df(\mathbf{X})}{dx_{ij}} \right\} = \left\{ \frac{d \sum_{k=1}^n \sum_{m=1}^n x_{km} a_{mk}}{dx_{ij}} \right\} = \{a_{ji}\} \quad (3.19)$$

Hence:

$$\frac{d \text{tr}(\mathbf{X}\mathbf{A})}{d\mathbf{X}} = \mathbf{A}^T \quad (3.20)$$

Further details on the trace operator and its properties and derivatives can be found in [39].

3.1.6. The Frobenious Norm

Let \mathbf{A} be a square matrix. Then the Frobenius [48] matrix norm is

$$\|\mathbf{A}\|_F = \sqrt{\sum_{i=1}^m \sum_{j=1}^n |a_{ij}^2|} = \sqrt{\text{tr}(\mathbf{A}\mathbf{A}^H)} \quad (3.21)$$

where \mathbf{A}^H is the conjugate transpose. The derivative of the Frobenius norm is

$$\frac{\partial f}{\partial \mathbf{X}} \|\mathbf{X}\|_F^2 = \frac{\partial \text{tr}(\mathbf{X}\mathbf{X}^H)}{\partial \mathbf{X}} = 2\mathbf{X} \quad (3.22)$$

3.1.7. Frobenious Inner Product

The Frobenius inner product is an operation which takes two matrices and outputs a scalar. It is often denoted as $\langle \mathbf{A}, \mathbf{B} \rangle_F$. The matrices \mathbf{A} and \mathbf{B} must have the

same dimensions, but they are not restricted to square. Let $\mathbf{A} \in \mathbb{R}^{n \times m}$ and $\mathbf{B} \in \mathbb{R}^{n \times m}$. Then the Frobenius Inner product is:

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i,j} \overline{a_{i,j}} a_{i,j} \quad (3.23)$$

Where the bar denotes the complex conjugate. Explicitly the sum is:

$$\begin{aligned} \langle \mathbf{A}, \mathbf{B} \rangle_F &= \overline{a_{11}} b_{11} + \overline{a_{12}} b_{12} + \cdots + \overline{a_{1m}} b_{1m} \\ &\quad + \overline{a_{21}} b_{21} + \overline{a_{22}} b_{22} + \cdots + \overline{a_{2m}} b_{2m} \\ &\quad \vdots \\ &\quad + \overline{a_{n1}} b_{n1} + \overline{a_{n2}} b_{n2} + \cdots + \overline{a_{nm}} b_{nm} \end{aligned} \quad (3.24)$$

The Frobenius inner product can also be seen as the product of the two matrices vectorised, denoted as $\text{vec}(\mathbf{A})$

$$\text{vec}(\mathbf{A}) = \begin{pmatrix} a_{11} \\ a_{12} \\ \vdots \\ a_{21} \\ a_{22} \\ \vdots \\ a_{nm} \end{pmatrix}, \quad \text{vec}(\mathbf{B}) = \begin{pmatrix} b_{11} \\ b_{12} \\ \vdots \\ B_{21} \\ B_{22} \\ \vdots \\ b_{nm} \end{pmatrix} \quad (3.25)$$

$$\overline{\text{vec}(\mathbf{A})}^T \text{vec}(\mathbf{B}) = \begin{pmatrix} \overline{a_{11}} & \overline{a_{12}} & \cdots & \overline{a_{21}} & \overline{a_{22}} & \cdots & \overline{a_{nm}} \end{pmatrix} \begin{pmatrix} b_{11} \\ b_{12} \\ \vdots \\ b_{21} \\ b_{22} \\ \vdots \\ b_{nm} \end{pmatrix} \quad (3.26)$$

Then, it is clear that

$$\langle \mathbf{A}, \mathbf{B} \rangle = \overline{\text{vec}(\mathbf{A})}^T \text{vec}(\mathbf{B}) \quad (3.27)$$

Another neat property of the Frobenius inner product is that it is equal to the

trace of the matrix product:

$$\langle \mathbf{A}, \mathbf{B} \rangle_F = \sum_{i,j} \bar{\mathbf{A}}_{i,j} \mathbf{B}_{i,j} = \text{tr}(\bar{\mathbf{A}}^T \mathbf{B}) \quad (3.28)$$

3.1.8. Orthogonal set

Let \mathbf{V} be a vector space with a defined inner product. A orthogonal set is denoted as non zero vectors $v_1, v_2, \dots, v_k \in \mathbf{V}$ orthogonal to each other, i.e., $\langle v_i, v_j \rangle = 0$ for $i \neq j$.

If all the vectors are of unit norm, $\|v_i\| = 1 \quad \forall v_i \in \mathbf{V}$, then v_1, v_2, \dots, v_k form an orthonormal group.

3.1.9. Orthogonal group

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square matrix. If

$$\mathbf{A}^T \mathbf{A} = \mathbf{A} \mathbf{A}^T = I \quad (3.29)$$

then the matrix \mathbf{A} is said to be orthogonal. The set of $n \times n$ orthogonal matrices form the [Orthogonal group](#) ($O(n)$). An orthogonal matrix \mathbf{A} has its inverse equal to its transpose.

$$\mathbf{A}^T = \mathbf{A}^{-1} \quad (3.30)$$

Further, it can be shown that any orthogonal square matrix has a determinant of either +1 or -1:

$$1 = \det I = \det A^T A = \det A \det A^T = (\det A)^2 \implies \det(A) = \pm 1 \quad (3.31)$$

If the determinant is equal to +1, the matrix is in the [Special Orthogonal group](#) ($SO(n)$).

3.2. Eigendecomposition

Let $\mathbf{A} \in \mathbb{R}^{n \times n}$ be a square matrix. Any number λ is denoted an eigenvalue of \mathbf{A} if there exist a non-zero vector \vec{u} such that

$$\mathbf{A} \vec{u} = \lambda \vec{u} \quad (3.32)$$

The vector \vec{u} associated with the eigenvalue is referred to as an eigenvector. The set of all the eigenvectors associated with λ is referred to as the eigenspace. Ge-

ometrically, Eq. 3.32 says that under the transformation of \mathbf{A} , the eigenvectors only change magnitude or sign, but does not change direction. The orientation of $\mathbf{A}\vec{u}$ is the same as \vec{u} . Thus the eigenvalue λ can be viewed as how much the vector is scaled.

To find the eigenvalues, start with Eq. and insert the identity matrix \mathbf{I}_n . 3.32:

$$\mathbf{A}\vec{u} - \lambda\mathbf{I}_n\vec{u} = 0 \quad (3.33)$$

Rearrange

$$\vec{u}(\mathbf{A} - \lambda\mathbf{I}_n) = 0 \quad (3.34)$$

Hence the equation to be solved is

$$\mathbf{A} - \mathbf{I}_n\lambda = 0 \quad (3.35)$$

This is a linear system which is singular if and only if the determinant is zero. $\det(\mathbf{A} - I\lambda)$ is a polynomial of degree n , in which the roots is the The Fundamental Theorem of Algebra [9] implies that there are exactly n roots, including complex roots and repetition.

Let the matrix \mathbf{U} consist of the eigenvectors of \mathbf{A} joined together in a matrix, where each column of \mathbf{U} is an eigenvector of \mathbf{A} . Combine the eigenvalues in a diagonal matrix denotes as Λ . Then Eq. 3.32 can be written as

$$\mathbf{A}\mathbf{U} = \mathbf{U}\Lambda \quad (3.36)$$

or equivalently as

$$\mathbf{A} = \mathbf{U}\Lambda\mathbf{U}^{-1} \quad (3.37)$$

Eq. 3.37 is referred to as the eigendecomposition of \mathbf{A} .

There are several limitations, as only diagonalizable matrices can be factorized in this way. In addition, the process only works for square matrices.

3.3. Singular Value Decomposition

The **Singular Value Decomposition (SVD)** is a matrix factorization into three matrices. Let $\mathbf{A} \in \mathbb{R}^{m \times n}$, either complex or real. Then the full SVD of \mathbf{A} is:

$$\underbrace{\mathbf{A}}_{m \times n} = \underbrace{\mathbf{U}}_{m \times m} \times \underbrace{\mathbf{\Sigma}}_{m \times n} \times \underbrace{\mathbf{V}^*}_{n \times n} \quad (3.38)$$

Where $\mathbf{U} \in \mathbb{R}^{m \times n}$ is a complex unitary matrix, $\mathbf{\Sigma} \in \mathbb{R}^{n \times n}$ is a rectangular diagonal matrix with non-negative real numbers on the diagonal, and $\mathbf{V} \in \mathbb{R}^{n \times n}$ is a complex unitary matrix. If \mathbf{A} is Real, then \mathbf{U} and \mathbf{V} is guaranteed to be real orthogonal matrices [48], and SVD is denoted as $\mathbf{U}\mathbf{\Sigma}\mathbf{V}^*$.

The matrix $\mathbf{\Sigma}$ is a diagonal matrix with entries $\sigma_i = \mathbf{\Sigma}_{ii}$ and is uniquely determined by \mathbf{A} , and is referred to as the singular values of \mathbf{A} . The number of non-zero singular values is equal to the $\text{rank}(\mathbf{A})$. The Columns of \mathbf{U} and \mathbf{A} is referred to as the left- and right-singular of \mathbf{A} respectively. The matrices form two orthonormal bases $[u_1, \dots, u_m]$ and $[v_1, \dots, v_n]$ and can also be expressed as:

$$\mathbf{A} = \sum_{i=1}^r \sigma_i u_i v_i^*, \quad \text{where } r = \min\{m, n\} = \text{Rank}(\mathbf{A}) \quad (3.39)$$

σ is magnitude of the vector, u is the direction. For the rest of the thesis, the superscript $*$ will be swapped out with the T for ease of reading.

The procedure to find \mathbf{U} and \mathbf{V} is done through eigen-decomposition of $\mathbf{A}^T \mathbf{A}$ and $\mathbf{A} \mathbf{A}^T$:

$$\mathbf{A}^T \mathbf{A} = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \quad (3.40)$$

$$= \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (3.41)$$

\mathbf{U} is orthogonal:

$$= \mathbf{V}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{V}^T \quad (3.42)$$

$$= \mathbf{V}\mathbf{\Sigma}^2 \mathbf{V}^T \quad (3.43)$$

\mathbf{V} is given by the eigen-decomposition of $\mathbf{A}^T \mathbf{A}$. \mathbf{V} is an orthogonal matrix, because $\mathbf{A}^T \mathbf{A}$ is symmetric. To find \mathbf{U} :

$$\mathbf{A} \mathbf{A}^T = (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T)^T (\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T) \quad (3.44)$$

$$= \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \mathbf{V}\mathbf{\Sigma}\mathbf{U}^T \quad (3.45)$$

\mathbf{V} is orthogonal:

$$= \mathbf{U}\mathbf{\Sigma}\mathbf{\Sigma}\mathbf{U}^T \quad (3.46)$$

$$= \mathbf{U}\mathbf{\Sigma}^2 \mathbf{U}^T \quad (3.47)$$

The same logic applies here, $\mathbf{A} \mathbf{A}^T$ is symmetric, hence the eigendecomposition of $\mathbf{A} \mathbf{A}^T$ yields an orthogonal matrix of \mathbf{U} . To find $\mathbf{\Sigma}$, the square root of the eigenvalues of either $\mathbf{A}^T \mathbf{A}$ or $\mathbf{A} \mathbf{A}^T$ can be used.

For a matrix \mathbf{R} without full rank, the remaining columns or rows of vectors are set to zero, and denoted as the null space:

$$\mathbf{A} = \mathbf{U}\Sigma\mathbf{V}^T$$

$$= \underbrace{\begin{bmatrix} \mathbf{u}_1 & \mathbf{u}_2 & \dots & \mathbf{u}_r \end{bmatrix}}_{\text{Col } \mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{u}_{r+1} & \dots & \mathbf{u}_m \end{bmatrix}}_{\text{Nul } \mathbf{A}^T} \begin{bmatrix} \sigma_1 & 0 & \dots & 0 & 0 & \dots & 0 \\ 0 & \sigma_2 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & \sigma_r & 0 & \dots & 0 \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \\ \dots & & & & & & \\ 0 & 0 & \dots & 0 & 0 & \dots & 0 \end{bmatrix} \left. \begin{array}{l} \left[\begin{array}{c} \mathbf{v}_1^T \\ \mathbf{v}_2^T \\ \dots \\ \mathbf{v}_r^T \\ \mathbf{v}_{r+1}^T \\ \dots \\ \mathbf{v}_n^T \end{array} \right] \\ \left. \begin{array}{l} \text{Row } \mathbf{A} \\ \text{Nul } \mathbf{A} \end{array} \right\} \end{array} \right\} \quad (3.48)$$

3.4. Jacobian matrix

Consider a functions $f : \mathbb{R}^n \rightarrow \mathbb{R}^m$ Then the Jacobian is defined as the all the partial derivatives for each function w.r.t each input. The Jacobian \mathbf{J} is of dimensions $m \times n$ and is defined as

$$\mathbf{J} = \begin{pmatrix} \frac{\partial f}{\partial x_1} & \dots & \frac{\partial f}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla^T f_1 \\ \vdots \\ \nabla^T f_m \end{pmatrix} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \dots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_m}{\partial x_1} & \dots & \frac{\partial f_m}{\partial x_n} \end{pmatrix} \quad (3.49)$$

The Jacobian is often referred to as the gradient of f , or simply ∇f .

3.5. Matrix differentials

3.5.1. Differentials

Before going in to matrix calculus; a notion of the differential is needed. The differential allows one to calculate gradients Consider a differentiable vector function $f(x)$ and the first order Taylor expansion:

$$f(x + dx) = f(x) + f'(x)dx + O(\|x\|^2) \quad (3.50)$$

Where $df(x; dx) = f'(x)u$ is denoted as the differential of f at x with perturbation of dx . Another way of representing the differential would be with partial derivatives:

$$df(x; dx) = (\mathbf{D}f(x))u \quad (3.51)$$

where $\mathbf{D}_j f_i(x)$ is the partial derivatives of f_i w.r.t., the j 'th coordinate at x and its transpose is the gradient of f at x :

$$\nabla f(x) = (\mathbf{D}f(x))^T \quad (3.52)$$

For a nested function, the chain rule can be utilized to find the differential: Let $dx = g \circ f$, then the differential of dx at x is:

$$\begin{aligned} dh(x; dx) &= (\mathbf{D}h(x))dx \\ &= (\mathbf{D}g(f(x)))(\mathbf{D}f(x))dx \\ &= (\mathbf{D}g(f(x)))df(x; dx) \\ &= dg(f(x); df(x; dx)) \end{aligned} \quad (3.53)$$

Extending the results for a matrix function: Let $\mathbf{F} : \mathbb{R}^{(m \times p)} \rightarrow \mathbb{R}^{(n \times q)}$:

$$\text{vec}F(\mathbf{X} + d\mathbf{X}) = \text{vec}F(\mathbf{X}) + \mathbf{F}'(\mathbf{X})\text{vec}(d\mathbf{X}) + O(\|d\mathbf{X}\|^2) \quad (3.54)$$

where the differential of \mathbf{F} at $d\mathbf{X}$ is denoted as:

$$\text{vec}dF(\mathbf{X} : d\mathbf{X}) = \mathbf{F}'(C)\text{vec}(d\mathbf{X}) \quad (3.55)$$

Note that even though the differential and the partial derivatives are related, they are different objects. Let $dg = dg(\mathbf{X}; d\mathbf{X})$ be the differential of g . Then dg is always defined if g is defined. The differential can take matrix inputs and map to a space of matrices. On the other hand, the partial derivative is only defined if g has a scalar co-domain [19, 30]. The differential is only used as a step to calculate the derivatives and does not require implementation. The main purpose is to find the expression for the partial derivatives.

Chapter 4.

Artificial Neural Networks

The first sections of this chapter are to a large extent based on the theoretical sections of the preliminary studies[14].

Within the field of deep learning, [Artificial Neural Network \(ANN\)](#) is in the center. ANN is a collection of algorithms that is loosely inspired by the neurological cells in the human brain. There are uncountable variants of ANNs, and multiple ways to train them. The following sections give an overview of the foundations of ANNs and the architecture of a standard ANN.

4.1. The Artificial Neuron

The human brain consists of more than 86 billion neurons that are interconnected as a massive network. [18]. Each of these biological neurons has the ability to propagate an electrical signal to a connection point, which is referred to as a synapse. The electrical signal is further propagated if and only if it is above a certain threshold in strength. This way, information is transferred in the brain, through a massive network of neurons. This neuroscientific model is the inspiration for ANNs. The Idea was incepted by McCulloch and Pitts in 1943 [33], and the first artificial neuron is referred to as the McCulloch-Pitts neuron. The modern artificial neuron has been further developed, and will be presented in the following section.

An artificial neuron is a simple mathematical function that acts as the fundamental building block of each neural network. Each neuron takes input $x \in \mathbb{R}^n$ and outputs a scalar $y \in \mathbb{R}$. Each element $[x_1, x_2 \dots, x_n]$ in x is multiplied with an associated weight $[w_1, w_2, \dots, w_n]$ from $w \in \mathbb{R}^n$. The values in the weight vector indicate the importance of the different elements in x . The weighted data is

summed up, and a bias is added.

$$z = \sum_{i=1}^n x_i w_i + b \quad (4.1)$$

This linear combination is thereafter non-linear transformed with an activation function. The result is an output for each neuron, which is often referred to as the activation of the neuron.

$$\hat{y} = f(z) \quad (4.2)$$

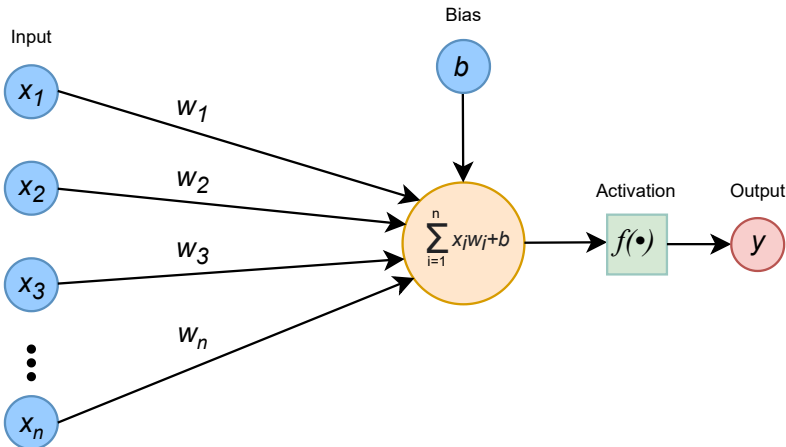


Figure 4.1.: Illustration of an artificial neuron. The input is a vector of length n , which is element-wise multiplied with the associated weights w , and summed with a bias. An activation function f is applied, which produces a scalar activation y .

The activation function can be seen as analogous to the threshold level in the biological neuron. The signal is only transmitted if it is above a given value. The original McCulloch-Pitts neuron used the step function in Eq.4.3.

$$f(x) = \begin{cases} 1 & \text{if } x > \theta \\ 0 & \text{if } x \leq \theta \end{cases} \quad (4.3)$$

More modern architectures usually apply gradually changing types of activation functions. Usual characteristics are monotonically increasing, continuous, and differentiable with a sigmoid shape. For a long time, the most common activation function was the logistic sigmoid function which squashes the input to the interval of $(0, 1)$. The function is illustrated in Figure4.2.

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \quad \sigma'(x) = \sigma(x)(1 - \sigma(x)) \quad (4.4)$$

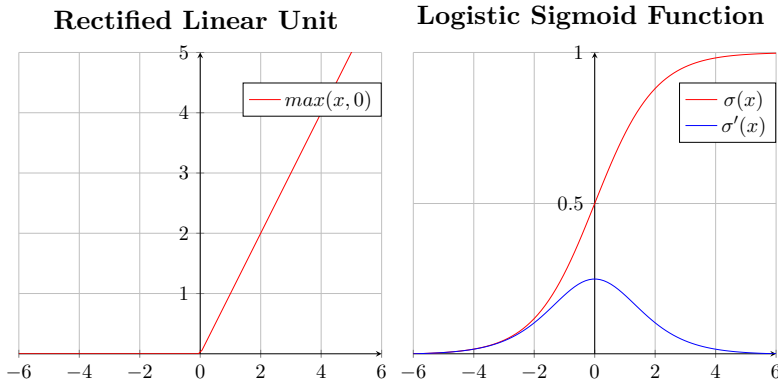


Figure 4.2.: Rectified Linear Unit and the Logistic Sigmoid

The most common activation function in modern neural networks is the rectified linear unit (ReLU) and its variants. ReLU is far more efficient than sigmoid [40]. ReLU is simply a linear function for values above zero, otherwise the value is set to zero, as shown in Eq. 4.5. For an introduction and comparison of different activation functions, the interested reader is guided to [40].

$$\text{ReLU}(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad \frac{d\text{ReLU}(x)}{dx} = \begin{cases} 1 & \text{if } x > 0 \\ 0 & \text{if } x \leq 0 \end{cases} \quad (4.5)$$

The artificial neuron is a linear function which is transformed non-linearly with an activation function. Multiple neurons can be combined in a layer, which is connected. Each layer takes the previous layers activations as their input, and passes their activations on to the next. This is the basis of the artificial neural network, and if every neuron is connected layer wise, it is a fully connected feed forward network.

4.2. Fully Connected Feed Forward Network

A feed-forward network is the foundation for the various techniques in deep learning. The network f tries to estimate a true function f^* . Due to the non-linearity induced by the activation function, theoretically, a sufficiently large enough network can represent any function f^* [13, p. 192]. The network f is denoted as

$f(x, \theta)$, where θ is the parameters of the network, i.e., the weights and biases and x is the input data.

A fully connected network consists of three parts: The input layer, the hidden layers, and the output layer. The general architecture is illustrated in Figure 4.3.

The input layer is a vector of size n , and there are two hidden layers. The output layer is also a vector of size k . Each layer is a stack of neurons. Each neuron has n weights w and a bias b . The neurons in the first layer $f^{(1)}$ are connected to the neurons in the second layer $f^{(2)}$. Each connection between the neurons from the layer $f^{(l-1)}$ to $f^{(l)}$ represents the weights multiplied with the activations from layer $l - 1$ with calculates the input to the l . The ultimate layer constructs a vector y of size k , such that $\dim y = \dim f^*$.

Each layer in the network processes a vector $x = [x_1, x_2, \dots, x_n]^T$, and performs the operation described in detail in subsection 4.1 with the weights w and biases b in each neuron. These are the trainable parameters of the network, denoted $\theta^{(l)} = \{W^{(l)}, b^{(l)}\}$. Each layer can thus be represented as a function:

$$f^{(l)}(x^{(l-1)}; \theta^{(l)}) = g^{(l)}(W^{(l)}x^{(l-1)} + b^{(l)}), \quad l = 1, 2, \dots, L, \quad (4.6)$$

Where $W^{(l)} \in \mathbb{R}^{n_{l-1} \times n_l}$, $b^l \in \mathbb{R}^{n_l}$, n_l refers to the number of neurons in layer l , $g(\cdot)$ is the activation function used in the neuron, and L is the number of layers in total, referred to the depth of the network.

As each network is a function that propagates its output to the following layer, the network can be written as nested functions:

$$f(x, \theta) = f^{(L)} \circ f^{(L-1)} \circ f^{(L-2)} \circ \dots \circ f^{(2)} \circ f^{(1)}(x) \quad (4.7)$$

4.3. Convolutional Neural Network

A feed forward networks performance is limited when working with higher-dimensional data exceeding one dimension. Consider an 2D image being the input of a feed forward network. Then the input will be flattened into a one dimensional vector where each pixel is an entry of the given vector. This causes the spatial information of the image to be lost. The **Convolutional Neural Network (CNN)** [23] alleviates this problem by maintaining the spatial information between the pixels with the use of the convolutional operator.

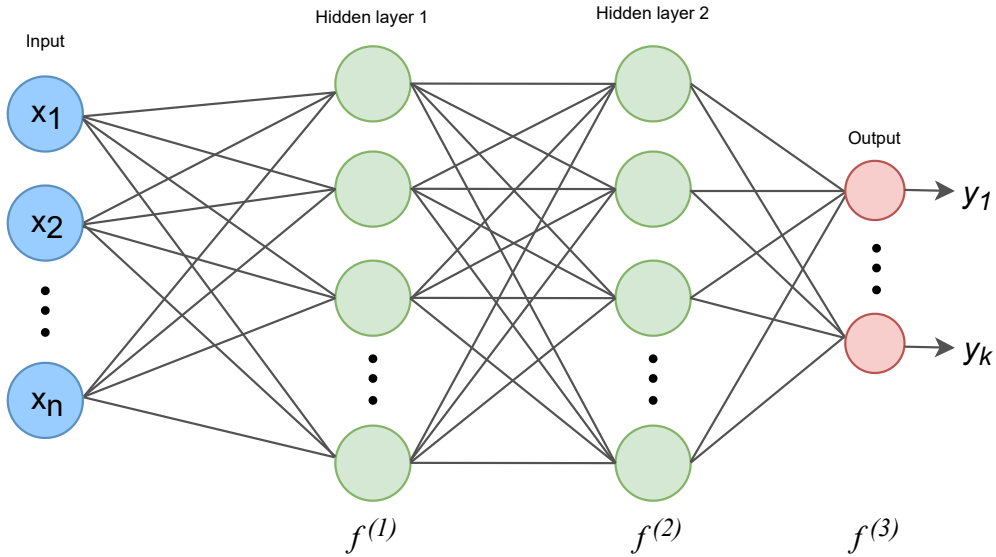


Figure 4.3.: Illustration of a fully connected neural network with two hidden layers. The network has an input vector of $x = [x_1, x_2, \dots, x_n]^T$, which propagates through the layers and produces the output vector $\hat{y} = [y_1, y_2, \dots, y_k]^T$

4.3.1. Convolutional Operator

The *convolutional* operator can be seen as passing a filter over an image, producing an output. The filter is referred to as the *kernel*, and the output is often referred to as a feature map. Let $\mathbf{I}(i, j)$ be an image, and $\mathbf{K}(i, j)$ be the kernel. Then the discrete convolution over the image is denoted as:

$$\mathbf{S}(i, j) = \mathbf{I}(i, j) * \mathbf{K}(i, j) = \sum_m \sum_n \mathbf{I}(i - m, j - n) \mathbf{K}(m, n) \quad (4.8)$$

where $*$ denotes the convolutional operator.

A more intuitive explanation is that for each pixel in the input image \mathbf{I} , the border-pixels are element-wise multiplied to the kernel. Visually, this can be imagined as the kernel hovers over the pixel in focus, and the elements that match up is multiplied and summed up, resulting in the output pixel.

The convolutional operator can be altered by applying *padding* or *stride*. Padding refers to artificial bordering around an image, whilst stride is how many pixels that are hovered over. For an more in-depth explanation of the convolutional operator and the different parameters, the interested reader is guided to the supplementary material [A.1](#).

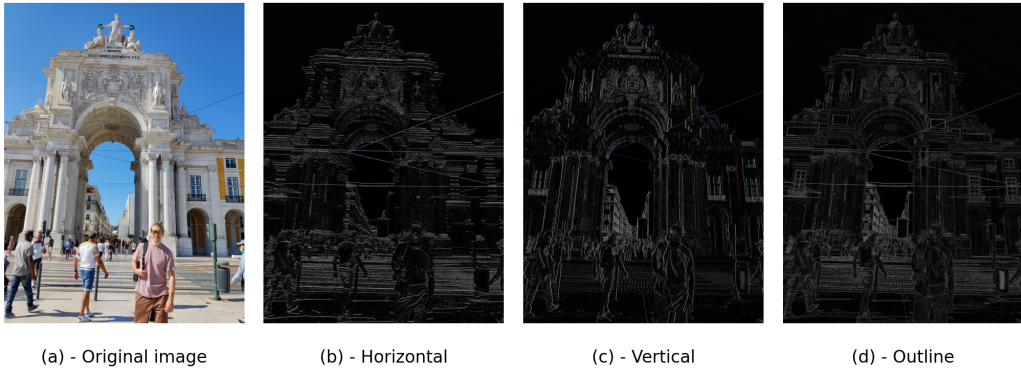


Figure 4.4.: An image convolved with the three kernels displayed in Eq. 4.10. The last is the Laplacian kernel which is frequently used for edge detection.

4.3.2. The Convolutional Layer

Usually, the input of the convolutional layer is 2D, and the operation can be denoted as:

$$f^{(l)}(x^{(l-1)}; \theta^{(l)}) = g^{(l)}(\mathbf{K}^{(l)} * \mathbf{X}^{(l-1)} + b^{(l)}) \quad (4.9)$$

where $\theta^{(l)} = \{\mathbf{K}^{(l)}, b^{(l)}\}$. Let \mathbf{X} be a 2D image, with height and width denoted as $h \times w$, then $\mathbf{X}^{(0)} \in \mathbb{R}^{h \times w}$ and the kernel has dimensions of $\mathbf{K} \in \mathbb{R}^{(h_l \times w_l)}$. However, there are variations of the exact implementations due to e.g., RGA images or parallelization of the batches (See [13, p. 347-358] for details).

To show the potency of the convolutional operator, consider the kernels:

$$\mathbf{K}_H = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix}, \quad \mathbf{K}_V = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}, \quad \mathbf{K}_O = \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix} \quad (4.10)$$

Figure 4.4 shows the effect of the kernels convolved over an image of the city center in Lisbon. The kernels displayed are the horizontal, vertical, and the outline kernel, respectively. \mathbf{K}_H and \mathbf{K}_V extract the horizontal and vertical edges of the photo, whilst the outline extracts the edges. The collection of the resulting images can be seen as a feature map, where the characteristics associated with edges are emphasized.

The convolutional layer consists of l kernels of dimension $k \times k$. Let the input be an image with c channels, i.e., of dimension $c \times m \times m$. Each kernel is then

convoluted over the c matrices of dimension $m \times m$ and summed up, resulting in l outputs with the dimensions $n \times n$, where n can be calculated from Eq. 4.11

$$n = \left\lceil \frac{m - k + 2P}{S} \right\rceil + 1 \quad (4.11)$$

A CNN typically consists of several convolutional layers before the penultimate layer being a fully connected linear layer. Note that the elements in the kernels are the parameters of any convolutional layer. The last layer of a CNN is typically a fully connected layer, outputting a vector of size \mathbb{R}^d .

In addition to the ability to retain the relationship between the image pixels, CNN has other beneficial properties for image processing. Consider the RGB image used in 4.4, which is of size $3 \times 3000 \times 4000$. A feed-forward layer with one weight per pixel would require $3.6 * 10^7$ million parameters, whilst a convolutional layer with a filter size of 3×3 requires 9 per kernel. In addition, the number of floating operations in the feed-forward layer is dramatically reduced using a convolutional layer.

4.4. Learning the Parameters - intuition

The learning of the parameters happens during a process which is referred to as the training of the network. During supervised training, each datavector will have an associated label. The data set is hence of the form: $\mathbf{X} = \{x_i, y_i\}_{i=1}^N$, where x_i is a vector with the property $y_i = f^*(x_i)$, i.e., the goal of the network output. The dataset is usually divided into a *training set* and *test set*, which is 80% and 20% of the data respectively. The training set is used to learn the parameters, and the test set is used to evaluate the performance of the finished trained model.

The idea behind the training procedure is to compare $f(x_i; \theta) = \hat{y}$ with the associated label to x_i , i.e., y_i . The difference is then utilized to tune the weights to improve the estimate. The procedure is done by the use of a loss function which quantifies the difference between the estimate and the label. The algorithm for the weight-tuning is often a gradient based optimization algorithm. A commonly used loss function is the mean squared error:

$$L(\hat{y}_i, y_i) = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2 \quad (4.12)$$

Inserting the network $f(x; \theta)$

$$L(\hat{y}_i, y_i) = \frac{1}{n} \sum_{i=1}^n (f(x; \theta) - y_i)^2 \quad (4.13)$$

The only adjustable parameters are θ . Hence, the goal is to find the set of parameters θ which minimizes the loss. This is an unconstrained convex optimization problem, and is solved by applying an optimization method.

$$L(\hat{y}_i, y_i) = \frac{1}{n} \sum_{i=1}^n (f(x; \theta) - y_i)^2 \quad (4.14)$$

after applying an optimization routine, e.g., gradient descent, the parameter update rule becomes that of Eq. 4.15.

$$\theta = \theta - \mu \nabla_{\theta} L, \quad (4.15)$$

μ is the learning rate (step rate). Figure 4.5 illustrates how one iteratively through small steps can reach the minimum point by calculating the gradient and hence direction towards the minimum point. In practice however, the loss function is multidimensional with numerous global minima, making the training procedure substantially more difficult than illustrated in Figure 4.5.

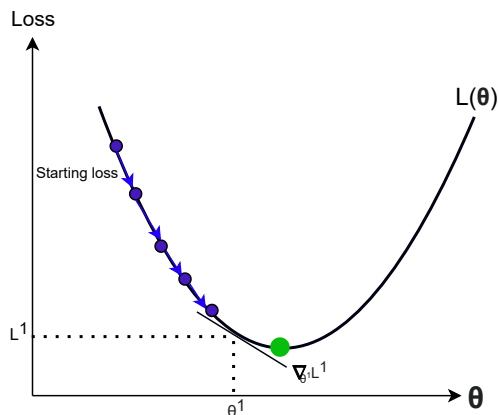


Figure 4.5.: Through tuning of the parameters with the Equation 4.15, the loss will iteratively reach the minimum point

4.4.1. Convergence Under Empirical Risk Minimization

Consider the network $f(x; \theta)$ to be able to give useful results, the set of parameters needs to be tuned such that $f(x; \theta) \approx f^*$. That is, the goal is for the model to perform well on unseen data, i.e., the ability to generalize. This can be guaranteed given that the principle of risk minimization is true:

$$\arg \min_{\theta} \frac{1}{N} \sum_{i=1}^N L(f(x^{(i)}, \theta), y^{(i)}) \quad (4.16)$$

As long as L and f are continuous, the gradients (or subgradients) can be used to learn the parameters. The general procedure is to use the backpropagation algorithm.

4.5. Backpropagation

The following sections are derived from [10, 24]. The ability to learn the models is based upon the gradients of the loss function. Recall that the network with depth k can be represented as

$$f = f^{(k)} \circ f^{(k-1)} \circ f^{(k-2)} \circ \dots \circ f^{(2)} \circ f^{(1)} \quad (4.17)$$

With the parameters given layer-wise as $\theta = (\theta^{(k)}, \theta^{(k-1)}, \dots, \theta_1)$ and $f^{(i)}$ denotes the layer i . Each layer's output is the next layers input, i.e.;

$$x_i = f^{(i)}(x_{i-1}), \quad i = 1, 2, \dots, K \quad (4.18)$$

x_{i-1} is the input of the layer f^i , which produces the output x_i . x_0 is the input data for the first layer.

Borrowing notation from [19]: $L^{(i)} = L \circ f^{(k)} \circ \dots \circ f^{(i)}$ denotes the loss of the as a function of the layer x_{l-1} . Ay the use of the chain rule, backpropagation calculates the derivaties of the loss function w.r.t all of the parameters. This requires the calculation of the gradients with respect of the inputs as well. To find the gradients of L at layer l :

$$\frac{\partial L^{(l)}(x_{l-1}, y)}{\partial w_l} = \frac{\partial L^{(l+1)}(x_l, y)}{\partial x_l} \frac{\partial f^{(l)}(x_{l-1}, y)}{\partial w_l} \quad (4.19)$$

$$\frac{\partial L^{(l)}(x_{l-1}, y)}{\partial x_{l-1}} = \frac{\partial L^{(l+1)}(x_l, y)}{\partial x_l} \frac{\partial f^{(l)}(x_{l-1})}{\partial x_{l-1}} \quad (4.20)$$

where f includes the activation functions as well. The end goal is to calculate the expression for the gradient in Eq. 4.19 and update the weights.

4.6. Matrix Backpropagation

Matrix Back-propagation (MBP) [19] is the use of matrix calculus[39] to map between the partial derivatives $\frac{\partial L^{(l+1)}}{\partial x_i}$ and $\frac{\partial L^{(l)}}{\partial x_i}$ at two consecutive layers. Let f be a network of depth K , with the layer $f^{(K)}$ be expressed in term of matrices, i.e., $\mathbf{X}_k = f^{(K)}(\mathbf{X}_{K-1})$, where \mathbf{X}_{K-1} and \mathbf{X}_K are matrices. Let $\mathbf{Y} = \mathbf{X}_K$, $\mathbf{X} = \mathbf{X}_{K-1}$ and $f = f^{(k)}$:

Let f be a network of depth K , with the layer $f^{(K)}$ be expressed in term of matrices, i.e., $\mathbf{X}_k = f^{(K)}(\mathbf{X}_{K-1})$, where \mathbf{X}_{K-1} and \mathbf{X}_K are matrices. Let $\mathbf{Y} = \mathbf{X}_K$, $\mathbf{X} = \mathbf{X}_{K-1}$ and $f = f^{(k)}$:

$$\mathbf{Y} = f(\mathbf{X}) \quad (4.21)$$

4.6.1. Partial derivatives

Let f be a network of depth K , with the layer $f^{(K)}$ be expressed in term of matrices, i.e., $\mathbf{X}_k = f^{(K)}(\mathbf{X}_{K-1})$, where \mathbf{X}_{K-1} and \mathbf{X}_K are matrices. Let $\mathbf{Y} = \mathbf{X}_K$, $\mathbf{X} = \mathbf{X}_{K-1}$ and $f = f^{(k)}$:

$$\mathbf{Y} = f(\mathbf{X}) \quad (4.22)$$

The objective is to calculate

$$\frac{\partial L \circ f}{\partial \mathbf{X}} \quad (4.23)$$

The strategy is to take the Taylor expansion of the matrix functions $L \circ f(\mathbf{X})$ around $d\mathbf{X}$ and of $L(\mathbf{Y})$

$$L \circ f(\mathbf{X} + d\mathbf{X}) - L \circ f(\mathbf{X}) = \frac{\partial L \circ f}{\partial \mathbf{X}} : d\mathbf{X} + O(\|d\mathbf{X}\|^2) \quad (4.24)$$

$$L(\mathbf{Y} + d\mathbf{Y}) - L(\mathbf{Y}) = \frac{\partial L}{\partial \mathbf{Y}} : d\mathbf{Y} + O(\|d\mathbf{Y}\|^2) \quad (4.25)$$

where $\mathbf{A}:\mathbf{B}$ is the Frobenius inner product: $\mathbf{A}:\mathbf{B} = \langle \mathbf{A}, \mathbf{B} \rangle_F = \text{vec}(\mathbf{A})^T \text{vec}(\mathbf{B})$.

As $\mathbf{Y} = f(\mathbf{X})$, the differential of \mathbf{Y} is:

$$d\mathbf{Y} = df(\mathbf{X}; d\mathbf{X}) \quad (4.26)$$

and if the relation holds, then the first order Taylor expansions in Eq. 4.24 and Eq. 4.25 are equal, and hence

$$\frac{\partial L}{\partial \mathbf{Y}} : d\mathbf{Y} = \frac{\partial L \circ \mathbf{f}}{\partial \mathbf{X}} : d\mathbf{X} \quad (4.27)$$

The last relation is used to map the partial derivative on the l.h.s to the partial derivatives on the r.h.s. The procedure is as follows: Define a functional \mathcal{L} :

$$d\mathbf{Y} = \mathcal{L}(d\mathbf{X}) \triangleq df(\mathbf{X}; d\mathbf{X}) \quad (4.28)$$

Given the expression for $d\mathbf{Y}$, the relation in Eq. 4.27 holds, and the properties of the Frobenius inner product $\mathbf{A} : \mathbf{B} = \text{Tr}(\mathbf{A}^T \mathbf{B})$ is used to obtain the partial derivatives. Let \mathcal{L}^* be a non-linear adjoint operation, which has the property $a : (L)(b) = \mathcal{L}^*(a) : b$. Then

$$\frac{\partial L}{\partial \mathbf{Y}} : d\mathbf{Y} = \frac{\partial L}{\partial \mathbf{Y}} : \mathcal{L}(d\mathbf{X}) \triangleq \mathcal{L}^* \left(\frac{\partial L}{\partial \mathbf{Y}} \right) : d\mathbf{X} \Rightarrow \mathcal{L}^* \left(\frac{\partial L}{\partial \mathbf{Y}} \right) = \frac{\partial L \circ \mathbf{f}}{\partial \mathbf{X}} \quad (4.29)$$

Yielding the result of the partial derivative. (if this is unclear, study Eq. 4.27 and note that if $\frac{\partial L}{\partial \mathbf{Y}} : d\mathbf{Y} = \frac{\partial L \circ \mathbf{f}}{\partial \mathbf{X}} : d\mathbf{X} = \mathcal{L}^* \left(\frac{\partial L}{\partial \mathbf{Y}} \right) : d\mathbf{X}$.)

4.7. Backpropagation for a SVD layer

Consider a network with a layer $\mathbf{Y} = \mathbf{f}(\mathbf{X})$, where $\mathbf{f}(\mathbf{X})$ is the singular value decomposition:

$$\mathbf{f}(\mathbf{X}) = \mathbf{U}\mathbf{\Sigma}\mathbf{V}^T \quad (4.30)$$

where $\mathbf{Y} = \mathbf{f}(\mathbf{X}) \in \mathbb{R}^{(m,n)}$ and $m \geq n$, and \mathbf{U} and \mathbf{V} are orthogonal matrices. The following section will show how the partial derivatives $\frac{\partial L \circ \mathbf{f}}{\partial \mathbf{X}}$ can be expressed in terms of $\frac{\partial L}{\partial \mathbf{U}}$, $\frac{\partial L}{\partial \mathbf{\Sigma}}$, and $\frac{\partial L}{\partial \mathbf{V}}$. The strategy to find the partial derivatives is through the differentials $d\mathbf{U}$, $d\mathbf{\Sigma}$, $d\mathbf{V}$. Recall that for SVD, the matrix $\mathbf{\Sigma}$ is diagonal, hence the differential $d\mathbf{\Sigma}$ is diagonal.

The differential of \mathbf{Y} is:

$$d\mathbf{Y} = d\mathbf{U}\mathbf{\Sigma}\mathbf{V}^T + \mathbf{U}d\mathbf{\Sigma}\mathbf{V}^T + \mathbf{U}\mathbf{\Sigma}d\mathbf{V}^T \quad (4.31)$$

Divide by \mathbf{V}^T and by \mathbf{U} :

$$\mathbf{U}^T d\mathbf{X}\mathbf{V} = \mathbf{U}^T d\mathbf{U}\mathbf{\Sigma} + d\mathbf{\Sigma} + \mathbf{\Sigma}d\mathbf{V}^T\mathbf{V} \quad (4.32)$$

and for simplicity, define:

$$\mathbf{Q} = \mathbf{U}^T d\mathbf{X}\mathbf{V} \quad \mathbf{A} = \mathbf{U}^T d\mathbf{U}, \quad \mathbf{B} = \mathbf{V}^T d\mathbf{V} \quad (4.33)$$

where \mathbf{A} and \mathbf{B} are skew-symmetric matrices. Further

$$\mathbf{Q} = \mathbf{A}\boldsymbol{\Sigma} + d\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\mathbf{B}^T \quad (4.34)$$

Due to the diagonal nature of $d\boldsymbol{\Sigma}$ and the skew-symmetry of \mathbf{A} and \mathbf{B} , the matrices $\mathbf{A}\boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}\mathbf{B}$:

$$d\boldsymbol{\Sigma} = \mathbf{I}_k \circ \mathbf{Q} \quad (4.35)$$

where \mathbf{I}_k is the identity matrix of dimension k , and \circ is the Hadamard product. Now to find the differentials $d\mathbf{U}$, $d\mathbf{V}$, Due to orthogonality in \mathbf{U} and \mathbf{V} , the terms $\mathbf{A} = \mathbf{U}^T d\mathbf{U}$ and $\mathbf{B} = \mathbf{V}^T d\mathbf{V}$ are skew symmetric, such that the off diagonal part satisfies:

$$\bar{\mathbf{I}}_k \mathbf{Q} = \mathbf{A}\boldsymbol{\Sigma} + \boldsymbol{\Sigma}\mathbf{B}^T \quad (4.36)$$

Where $\hat{\mathbf{I}}_k$ is a zero-diagonal matrix with ones everywhere else. Rearranging Eq. 4.36 such that

$$\mathbf{A} = \bar{\mathbf{I}}\mathbf{Q}\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}\mathbf{B}^T\boldsymbol{\Sigma}^{-1} \quad (4.37)$$

Taking the transpose of \mathbf{A} :

$$\mathbf{A}^T = \boldsymbol{\Sigma}^{-1}(\bar{\mathbf{I}}\mathbf{Q})^T - \boldsymbol{\Sigma}^{-1}\mathbf{B}\boldsymbol{\Sigma} \quad (4.38)$$

By using the identity of $\mathbf{A} + \mathbf{A}^T = 0$, and rearrange:

$$\boldsymbol{\Sigma}^{-1}\mathbf{B}\boldsymbol{\Sigma} - \boldsymbol{\Sigma}\mathbf{B}^T\boldsymbol{\Sigma}^{-1} = (\bar{\mathbf{I}}\mathbf{Q})\boldsymbol{\Sigma}^{-1} + \boldsymbol{\Sigma}^{-1}(\bar{\mathbf{I}}\mathbf{Q})^T \quad (4.39)$$

Use the fact that $\mathbf{B}^T = -\mathbf{B}$ and pre and post multiply with $\boldsymbol{\Sigma}$:

$$\mathbf{B}\boldsymbol{\Sigma}^2 - \boldsymbol{\Sigma}^2\mathbf{B} = \boldsymbol{\Sigma}(\bar{\mathbf{I}}\mathbf{Q}) + (\bar{\mathbf{I}}\mathbf{Q})^T\boldsymbol{\Sigma} \quad (4.40)$$

Which is the same as:

$$\mathbf{F} \circ \mathbf{B} = \boldsymbol{\Sigma}(\bar{\mathbf{I}}_k\mathbf{Q}) + (\bar{\mathbf{I}}_k\mathbf{Q})^T\boldsymbol{\Sigma} \quad (4.41)$$

Where $\mathbf{F}_{ij} = \sigma_j^2 - \sigma_i^2$ and \circ is the Hadamard product. \mathbf{F} has a zero diagonal, hence the term $\bar{\mathbf{I}}$ actually redundant, but it is kept there for clarity. The same procedure can be repeated for \mathbf{B} , yielding:

$$\mathbf{F} \circ \mathbf{A} = (\mathbf{Q}\bar{\mathbf{I}}_k)\boldsymbol{\Sigma} + \boldsymbol{\Sigma}(\bar{\mathbf{I}}_k\mathbf{Q})^T \quad (4.42)$$

The last two expressions in addition to $d\Sigma = \mathbf{I}_k \circ \mathbf{Q}$ is then used to solve for $d\mathbf{U}$, $d\mathbf{V}$, and $d\Sigma$:

$$d\Sigma = \mathbf{I}_k \circ \mathbf{Q} = \mathbf{I}_k \circ \mathbf{U}^T d\mathbf{X} \mathbf{V} \quad (4.43)$$

$$\begin{aligned} \mathbf{F} \circ \mathbf{A} &= \mathbf{F} \circ (\mathbf{U}^T d\mathbf{U}) = (\bar{\mathbf{I}}\mathbf{Q})\Sigma + \Sigma(\bar{\mathbf{I}}\mathbf{Q})^T \Rightarrow \\ d\mathbf{U} &= \mathbf{U} \left(\mathbf{F}^{-1} \left((\bar{\mathbf{I}}_k \mathbf{Q})\Sigma + \Sigma(\bar{\mathbf{I}}_k \mathbf{Q})^T \right) \right) \\ d\mathbf{U} &= \mathbf{U} \left(\mathbf{F}^{-1} \circ \left((\bar{\mathbf{I}}_k \mathbf{U}^T d\mathbf{X} \mathbf{V})\Sigma + \Sigma(\bar{\mathbf{I}}_k \mathbf{U}^T d\mathbf{X} \mathbf{V})^T \right) \right) \end{aligned} \quad (4.44)$$

$$\begin{aligned} \mathbf{F} \circ \mathbf{B} &= \mathbf{F} \circ (\mathbf{V}^T d\mathbf{V}) = \Sigma(\bar{\mathbf{I}}\mathbf{Q}) + (\bar{\mathbf{I}}\mathbf{Q})^T \Sigma \Rightarrow \\ d\mathbf{V} &= \mathbf{V} \left(\mathbf{F}^{-1} \circ \left(\Sigma(\bar{\mathbf{I}}\mathbf{Q}) + (\bar{\mathbf{I}}\mathbf{Q})^T \Sigma \right) \right) \\ d\mathbf{V} &= \mathbf{V} \left(\mathbf{F}^{-1} \circ \left(\Sigma(\bar{\mathbf{I}}\mathbf{U}^T d\mathbf{X} \mathbf{V}) + (\bar{\mathbf{I}}\mathbf{U}^T d\mathbf{X} \mathbf{V})^T \Sigma \right) \right) \end{aligned} \quad (4.45)$$

With the differentials for \mathbf{U} , \mathbf{V} , Σ , it is now possible to find the expression for the $\frac{\partial \mathbf{f}}{\partial \mathbf{X}}$, which is the main goal:

First

$$\frac{\partial \mathbf{f}}{\partial \mathbf{X}} : d\mathbf{X} = \frac{\partial \mathbf{f}}{\partial \mathbf{U}} : d\mathbf{U} + \frac{\partial \mathbf{f}}{\partial \Sigma} : d\Sigma + \frac{\partial \mathbf{f}}{\partial \mathbf{V}} : d\mathbf{V} \quad (4.46)$$

$$\begin{aligned} \frac{d\mathbf{f}}{d\mathbf{X}} &= \mathbf{U} \left(\mathbf{F}^{-1} \circ \left[\mathbf{U}^T \frac{\partial \mathbf{f}}{\partial \mathbf{U}} - \frac{\partial \mathbf{f}^T}{\partial \mathbf{U}} \right] \Sigma \right) \mathbf{V}^T \\ &+ \mathbf{U} \left(\mathbf{I}_k \circ \frac{\partial \mathbf{f}}{\partial \Sigma} \right) \mathbf{V}^T \\ &+ \mathbf{U} \left(\Sigma \mathbf{F}^{-1} \circ \left[\mathbf{V}^T \frac{\partial \mathbf{f}}{\partial \mathbf{V}} - \frac{\partial \mathbf{f}^T}{\partial \mathbf{V}} \mathbf{U} \right] \right) \mathbf{V}^T \end{aligned} \quad (4.47)$$

This result will be used for the later chapters.

Chapter 5.

Transformations and coordinate frames

A rigid-body object's pose in 3D has 6 degrees of freedom (DOFs). Each object has a position that can be represented by its x, y, and z coordination in space, and each object can be rotated around the same axes, hence 6 DOFs. The most common representations of rotations are the Euler-angles, rotation matrices, and quaternions.

5.1. Rotation Matrix

Each rotation of a 3D object can be uniquely described by a rotation matrix. Let R be a special orthogonal (SO(3)) matrix with determinant equal to 1.

$$R = \begin{pmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{pmatrix} \quad (5.1)$$

Even if there are nine entries, because a rigid body 3D object only has three DOFs, only three of the entries can be chosen independently. As a result, each rotation matrix has six constraints. The following conditions must be satisfied for a 3D rotation matrix:

1. The unit norm condition:

$$r_{11}^2 + r_{21}^2 + r_{31}^2 = 1$$

$$r_{12}^2 + r_{22}^2 + r_{32}^2 = 1$$

$$r_{13}^2 + r_{23}^2 + r_{33}^2 = 1$$

2. The orthogonality condition

$$\begin{aligned} r_{11}r_{12} + r_{21}r_{22} + r_{31}r_{32} &= 0 \\ r_{12}r_{13} + r_{22}r_{23} + r_{32}r_{33} &= 0 \\ r_{11}r_{13} + r_{21}r_{23} + r_{31}r_{33} &= 0 \end{aligned}$$

The set of constraints can be summed up more compact as $R^T R = R R^T = I$, where I is the identity matrix. Furthermore, limiting the coordinate frames to be only right-handed frames, the constraint $\det(R) = 1$ is imposed. From this follows the fact that $R^{-1} = R^T$

The set of all 3×3 rotation matrices forms the special orthogonal group $SO(3)$. For rotations in 2D, i.e., the rotation matrices form $SO(2)$, which a subgroup of $SO(3)$. It can be shown that every $R \in SO(2)$ can be represented on the form [3]:

$$R = \begin{pmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \end{pmatrix} = \begin{pmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{pmatrix} \quad (5.2)$$

where $\theta = [0, 2\pi]$. The rotation matrices in $SO(2)$ represent planar orientations, whilst the 3D rotation matrices in $SO(3)$ represent spatial orientations. Note the discontinuity between when going from the directional limit from both sides, i.e., from 0 and 2π , which represents the same orientation in the limit.

5.1.1. Properties

The inverse of a rotation matrix is a rotation matrix

Let $R \in SO(3)$ be a rotation matrix. Then the inverse R^{-1} is a rotation matrix equal to the transpose of R . The proof is also straightforward. From the orthogonality constraint:

$$R^T R = I \implies R^{-1} = R^T \implies R R^T = I \quad (5.3)$$

R^{-1} is orthogonal. Further, to ensure special orthogonality:

$$\det(R)^T = \det(R) = 1 \quad (5.4)$$

Hence $R^{-1} = R^T$ is also a rotation matrix in $SO(3)$

The product of two rotation matrices is a rotation matrix

Let $R_1, R_2 \in SO(3)$, then the product $R_1 R_2$ is also a rotation matrix. The proof

is straight forward: The orthogonality constraint:

$$\begin{aligned} (R_1 R_2)^T (R_1 R_2) &= R_2 R_1^T R_1 R_2 \\ &= R_2^T R_2 I \\ &= I \end{aligned} \tag{5.5}$$

The special orthogonality constraint:

$$\det(R_1 R_2) = \det(R_1) \det(R_2) = 1 \tag{5.6}$$

Hence $R_1 R_2 \in SO(3)$ and is a rotation matrix.

5.1.2. Representing an Orientation

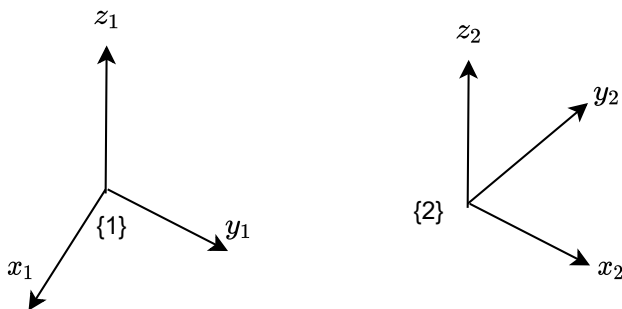


Figure 5.1.: Same space with two different orientations.

Assume that in Figure 5.1, there is a fixed space frame s aligned with 1. The orientations of the two frames 1 and 2 relative to s can be represented with the orientation matrices:

$$R_1 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, \quad R_2 = \begin{pmatrix} 0 & -1 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{5.7}$$

2 is obtained by rotating 1 by 90° .

5.1.3. Changing the Frame

A rotation matrix can be used to represent an orientation or to rotate a vector or a frame. Figure 5.2 illustrates how a frame 1 is rotated by an angle θ around a unit vector w , changing the frame to 2.

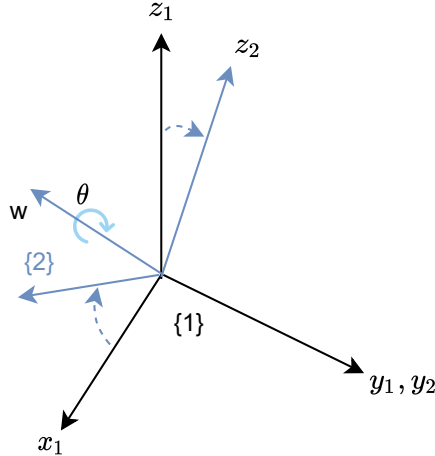


Figure 5.2.: A coordination frame with axes $[x_1, y_1, z_1]$ is rotated by θ about a unit axis w , aligned by $-y_1$. The orientation of the frame after the rotation is $[x_2, y_2, z_2]$ marked in blue. Figure adopted from [29]

The rotation matrix R can be written as a rotation operator, $R = Rot(w, \theta)$. Using Rodrigues formula [41]¹:

$$R_{1,2} = Rot(w, \theta) = I + \sin\theta[w] + (1 - \cos\theta)[w]^2 \quad (5.8)$$

where w is the skew-symmetric representation of a vector:

$$[w] = \begin{pmatrix} 0 & -w_3 & w_2 \\ w_3 & 0 & -w_1 \\ -w_2 & w_1 & 0 \end{pmatrix} \quad (5.9)$$

¹Although it was first discovered by Leonhard Euler in 1770 [11], it was rediscovered independently by Olinde Rodrigues in 1840 [41]. Cheng and Gupta proposed in 1989 the name “Euler’s finite rotation formula”[8].

5.2. Comparing Rotation Matrices

5.2.1. Angle error

The most common metric for comparing rotation matrices is the angle error. Let R_1 and R_2 denote two rotation matrices, and R_2^T be the transpose. Then the difference rotation is denoted as:

$$R_d = R_1 R_2^T \quad (5.10)$$

To calculate the distance between the rotations represented by R_1 and R_2 the angle of the difference rotation R_d is used. This can be retrieved by

$$\text{tr } R_d = 1 + 2 \cos \theta \quad (5.11)$$

and then using solving for θ :

$$\theta = \arccos \left(\frac{\text{tr } R - 1}{2} \right) \quad (5.12)$$

5.2.2. Geodesic Loss

Consider a sphere with a point on the top and bottom. The shortest absolute path would be right through the sphere, whereas the geodesic distance would be along the curvature. The geodesic loss is typically referred to as the shortest path between two points on a surface.

The geodesic distance between two groups in $\text{SO}(3)$ is denoted as:

$$d_g(\mathbf{R}_1, \mathbf{R}_2) = \frac{1}{\sqrt{2}} \left\| \log(\mathbf{R}_1^{-1} \mathbf{R}_2) \right\|_F \quad (5.13)$$

5.3. Camera Model

5.3.1. The Pinhole Camera Model

A camera is mathematically a projection of 3D points on a 2D image plane. To show this, consider a Euclidean coordinate system, and let the center of projection be the origin of the given system. Denote the axis $Z = f$ as the focal plane. The pinhole model then maps a point from the 3D world space $(X, Y, Z)^T$ to the 2D image coordinates $\left(\frac{fX}{Z}, \frac{fY}{Z} \right)^T$ as illustrated in Figure 5.3 The projection center

is referred to as the principal axis (or camera center), whereas the intersection between the principal axis and image plane is the principle point.

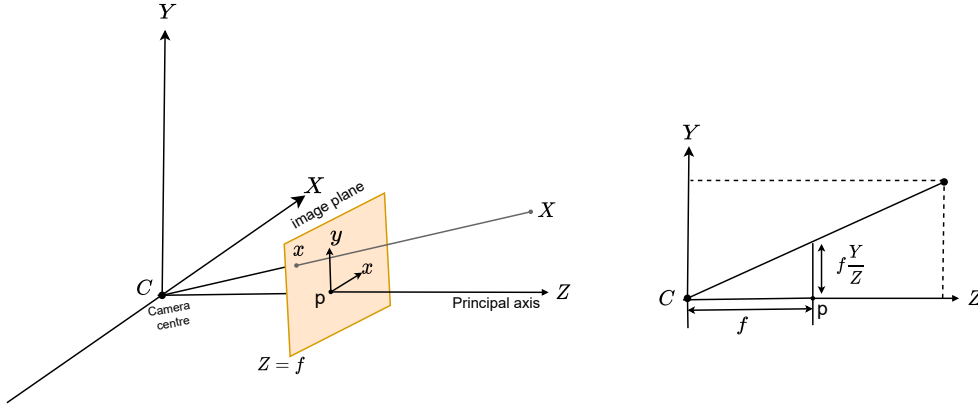


Figure 5.3.: The pinhole camera model: The left figure how a point X is mapped to the image plane on point x . The right figure shows how the height of the mapping is calculated through the triangle model, with the mapping of $y = f \frac{Y}{Z}$. Figure adapted from [16]

Assuming homogenous coordinates in the representation of both world and image points, the central projection can be expressed as a linear mapping:

$$\begin{pmatrix} fX \\ fY \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & 0 & 0 \\ 0 & f_y & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (5.14)$$

If there is an offset between the origin of coordinates in the image plane and camera center, then this needs to be accounted for:

$$\begin{pmatrix} fX + Zp_x \\ fY + Zp_y \\ Z \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & p_x & 0 \\ 0 & f_y & p_y & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \quad (5.15)$$

Where p_x, p_y is the principle point offset. The matrix on the right-hand side of Eq. 5.15 is the calibration matrix, usually denoted by K .

5.3.2. Camera Rotation and Translation

Up until now the coordinates X, Y, Z have been represented as camera coordinate, with the axes of the camera and world object coinciding. This is usually not the case, as objects in the world usually have their own axes. Denote the camera coordinate frame as $\mathbf{X}_{cam} = (X_{cam}, Y_{cam}, Z_{cam}, 1)^T$ and the world coordinate frame as X, Y, Z . As illustrated in Figure 5.4,

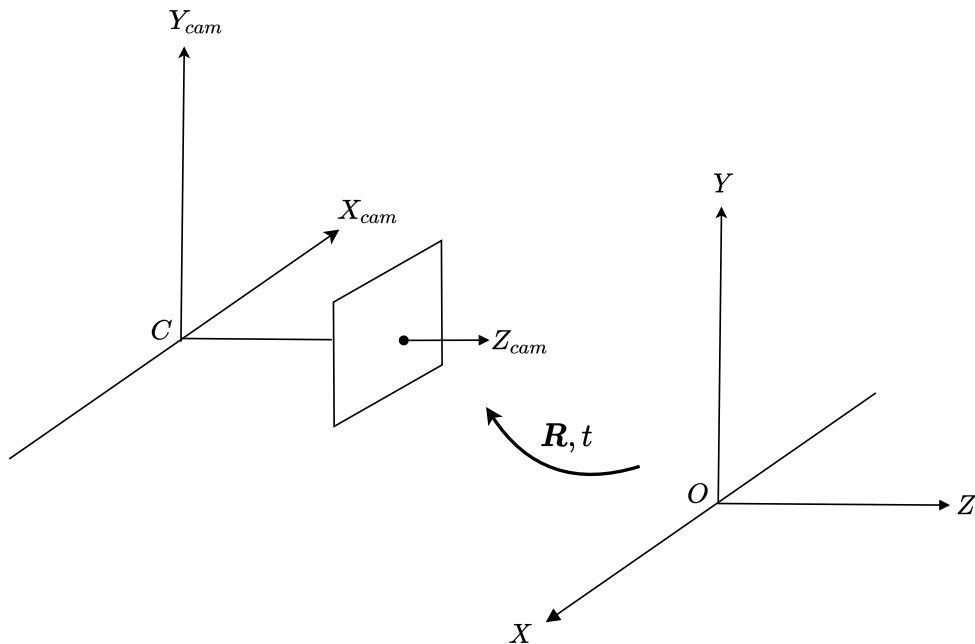


Figure 5.4.: An Euclidean transformation between the world and camera coordination frame. Figure adapted from [16]

$$\mathbf{X}_{cam} = [\mathbf{R} \quad t]\mathbf{X} \quad (5.16)$$

where $\mathbf{R} \in SO(3)$ is a rotation matrix, whereas $t \in \mathbf{R}^{3 \times 1}$ translation matrix. This leads to the general pinhole camera model:

$$\mathbf{x}_c = \mathbf{K}[\mathbf{R} \quad t]\mathbf{X}_w \quad (5.17)$$

The matrix \mathbf{K} is often referred to the intrinsic matrix, due to the fact that the entries hold parameters related to the camera. The external parameters is often

combined into a transformation matrix \mathbf{T} :

$$\mathbf{T} = \left(\begin{array}{c|c} \mathbf{R}_{3 \times 3} & t_{3 \times 1} \\ \hline 0_{1 \times 3} & 1 \end{array} \right) \quad (5.18)$$

The model can then be written as

$$\mathbf{x} = \begin{pmatrix} f_x & 0 & x_0 \\ 0 & f_y & y_0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \left(\begin{array}{c|c} \mathbf{R}_{3 \times 3} & t_{3 \times 1} \\ \hline t_{1 \times 3} & 1 \end{array} \right) \mathbf{X} \quad (5.19)$$

Where $\mathbf{P} = \mathbf{K}[\mathbf{R} \ t]$ is often referred to as the camera matrix. The pose of the camera relative to the world frame can be represented by the transformation. A valid transformation matrix is said to be in the special Euclidean group $\text{SE}(3)$.

For more details on the pinhole model, the reader is guided to [16].

5.4. Pose Estimation

The notion of pose estimation refers to estimating the relative pose between the object and a camera. Consider a scene with an object captured by a camera. Then the object will have its own coordination system, the camera will have its own, both separate from the world coordination system. Figure 5.5 illustrates the problem in the 3D space. T_{ij} between frame i, j denotes the transformation matrices between the objects. 3D pose estimation refers to the regression of the rotation of the object, whereas 6D pose estimation includes the translation as well. 6D pose estimation increases the complexity of the problem, the translation output requires other parametrization methods, which is out of scope for this thesis.

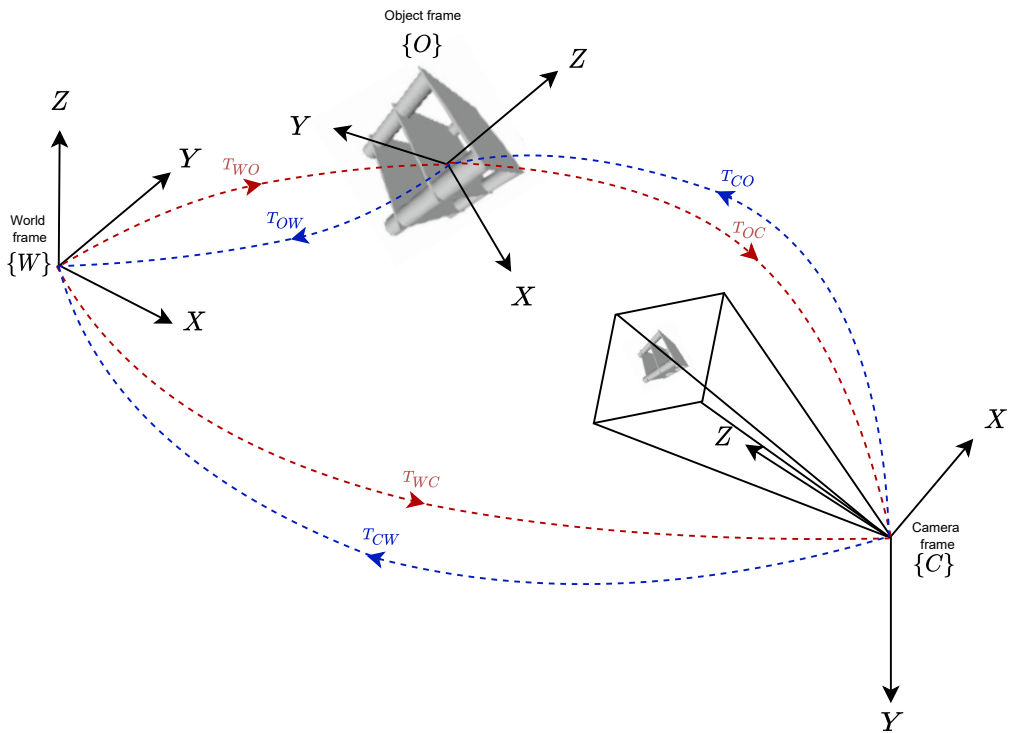


Figure 5.5.: The pose estimation problem: The matrix of interest is the transformation matrix T_{CO} between the camera and the object. The world coordination system is irrelevant to the task.

Chapter 6.

Mappings from Real Euclidean Space to Special Orthogonal Group

Consider a neural network with an output of $x \in \mathbf{R}^n$. The target is on a non-Euclidean space $SO(3)$ and hence the outputs needs to be transformed to $SO(3)$ to ensure a proper rotation matrix. The transformation is done with a mapping function: Define a differentiable function f such that $f : x \in \mathbf{R}^n \rightarrow f(x) \in Y$, i.e., a mapping from the output to the desired target. The network can then be trained with an arbitrary loss function and optimizer with back-propagation. The choice of such a mapping is a non-trivial task and has a significant effect on the model's performance. A thorough and satisfactory explanation of which properties make a good mapping is still lacking, but Brégier attempts to summarize the research on the area in [6].

6.1. Desired Properties

The desired properties for a differentiable mapping from $\mathbf{R}^n \rightarrow SO(3)$ is an active area of research. Zhou et al. argues that a good mapping should be surjective and be continuous, meaning that the right inverse $g : Y \rightarrow X$ should exists. Zhou et al. further demonstrates that for rotation matrices, all representations are discontinuis in the Euclidean space with four or fewer dimensions [54]. Brégier argues that this property is desirable, but nevertheless a a loose criterion. Brégier argues that the following function will satisfy the criterion:

$$R \in \mathcal{M}_{\exists, \exists} \mathbf{R} \rightarrow [M \quad \text{if} \quad M \in SO(3), I \text{ else}] \quad (6.1)$$

Such a mapping is likely to output the identity matrix often causing the gradients to be zero. For this reason, Brégier argues that stricter and a more in-depth analysis is needed. Hence Brégier establishes the desirable properties of a mapping [6] as follows:

f should be surjective, i.e., $f(X) = Y$. This is a necessary constraint to be able to estimate any arbitrary output $y \in Y$.

The function f needs to be differentiable The mapping f needs to be differentiable to be able to use back-propagation to learn the parameters of the network. Differentiability also implies that f should be continuous.

The manifold Y should be a connected smooth differentiable manifold. The ability to learn any arbitrary target $y \in Y$ necessitate the ability to differentiate f on every point in Y .

The remaining two properties are not strictly required, but will help with training and generalization [6]:

Full rank Jacobian: The Jacobian of f , i.e., $J = \nabla_x f$ should have rank equal to the dimensionality of Y . This property again related to the back-propagation process of the training procedure. If the property is satisfied, it is always possible to find an infinitesimal displacement to apply to x such that output $f(x)$ is closer to the target $y \in Y$. Further, it guarantees convergence of gradient descent to a global minimum of $x \rightarrow \mathcal{L}(f(x))$ where \mathcal{L} is a convex loss function admitting a lower bound [6].

Pre-images connectivity/convexity. The pre-image $f^{-1}(y)$ of any element $y \in Y$ should be connected, or better yet, convex. Brégier shows an example with a continuous network $h : a \in A \rightarrow x \in X$ producing an intermediate representation x from the input a . The machine learning task is to regress such that $y = f(h(a)) \in Y$ on the target manifold close to the target $y^* \in Y$. In a toy example, Brégier demonstrates that different outputs a_0, a_1 correspond roughly to the same target output y^* , whereas the intermediate representations x_0, x_1 might be distant from each other. The claim is that generalization requires the model to properly interpolate between training samples, which is impossible if x_0, x_1 is in the disconnected regions of $f^{-1}(y^*)$. The consequence is that some of the test representations interpolated by the network will not be mapped correctly to y^* .

The result is that a mapping satisfying the pre-image connectivity constraint prevents such a situation, effectively guaranteeing the existence of a network able to properly interpolate between training representations. It is however noted that this is hard to do in practice as training such a network is difficult [6].

6.2. Differentiable Mappings - Examples

6.2.1. Euler Angles

The Euler Angles are another way of representing rotations from $SO(3)$ by three successive rotations, each around a different axis. This leads to the Euler Theorem: “Any arbitrary rotation may be described by only three parameters“. These three parameters are the angles around the distinct axes. Each angle around the distinct axes has a different name and is represented by a distinct function. The convention is as follows:

1. A *yaw* is a counterwise rotation of α around the z-axis. The rotation matrix is given by:

$$R_z(\alpha) = \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (6.2)$$

2. A *pitch* is a counter-clockwise rotation of γ around the x axis. The rotation matrix is given by:

$$R_y(\beta) = \begin{pmatrix} \cos \beta & 0 & \sin \beta \\ 0 & 1 & 0 \\ -\sin \beta & 0 & \cos \beta \end{pmatrix} \quad (6.3)$$

3. A *roll* is a counter-clockwise rotation of β around the y-axis. The rotation matrix is given by:

$$R_x(\gamma) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \gamma & -\sin \gamma \\ 0 & \sin \gamma & \cos \gamma \end{pmatrix} \quad (6.4)$$

These three combined can result in any rotation in 3D. My composing the three rotations one rotation matrix can be obtained:

$$\begin{aligned} R &= R_z R_y R_x \\ &= \begin{pmatrix} \cos \alpha \cos \beta & \cos \alpha \cos \beta \cos \gamma - \cos \gamma \cos \alpha & \cos \alpha \cos \gamma + \cos \alpha \cos \gamma \cos \beta \\ \cos \beta \cos \alpha & \cos \alpha \cos \gamma + \cos \alpha \cos \beta \cos \gamma & \cos \gamma \cos \alpha \cos \beta - \cos \alpha \cos \gamma \\ -\cos \beta & \cos \beta \cos \gamma & \cos \beta \cos \gamma \end{pmatrix} \end{aligned} \quad (6.5)$$

As matrix multiplication, as well as rotations, are not commutative, the order

matters. There are six possible orderings, which would all yield different results.

The most dominant drawback of Euler angles is the phenomenon referred to as the *gimbal lock*. As Euler angles is a function that maps input $\mathbb{R}^3 \rightarrow SO(3)$, the derivative of this function is not guaranteed to have rank 3, and hence there are degenerate submanifolds where the function is many-to-one. This case is referred to as a gimbal lock, which causes the loss of one degree of freedom. The gimbal lock phenomenon causes Jacobian rank deficiency when it occurs. Furthermore, Euler angles does not satisfy the pre-images connectivity. There can exist multiple discrete pre-images for a given rotation, making the generalization hard.

Further, Euler angles are also discontinuous according to the definition of Zhou et al. [54]. This can be seen by the fact that rotations around one axis is equal for 0 and 2π . Hence Euler angles are unfit for regression of rotation matrices.

6.3. Quaternions

Quaternions are another way of representing rotations in 3D, which consists of 4 parameters. A quaternion is a hypercomplex number that alleviates the gimbal lock problem that the Eulers angles suffer from. Let q be quaternion representation:

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \end{bmatrix} = q_1 i + q_2 j + q_3 k + q_4 \quad (6.6)$$

where $q_{i \in 1,2,3,4}$ are real numbers, whereas i, j and k are imaginary units satisfying the following identity

$$i^2 = j^2 = k^2 = -1 \quad (6.7)$$

A quaternion can be seen as a scalar part $s = q_4 \in \mathbb{R}$ and a vector part $v = [q_1, q_2, q_3] \in \mathbb{R}^3$. Other notations of quaternions are:

$$\begin{aligned} q &= [s, v] \\ q &= [s, (q_1, q_2, q_3)] \end{aligned} \quad (6.8)$$

6.3.1. Properties

let q and q' be two quaternions. Then

$$q + q' = [s + s', v + v'] = (q_1 + q'_1)i + (q_2 + q'_2)j + (q_3 + q'_3)k \quad (6.9)$$

Which is associative and commutative.

The product of two quaternions q and q' is given by:

$$qq' = [ss' - vv', v \times v' + sv' + s'v] \quad (6.10)$$

The norm of a quaternion q is given as

$$\|q\| = \sqrt{q_1^2 + q_2^2 + q_3^2 + q_4^2} \quad (6.11)$$

The conjugate q^* of a quaternion is given as

$$q^* = -q_1i - q_2j - q_3k + q_4 \quad (6.12)$$

Then the following relations holds true:

$$qq^* = \|q\|^2 \quad (6.13)$$

And the inverse of a quaternion can be found by rearranging:

$$q^{-1} = \frac{q^*}{\|q\|^2} \quad (6.14)$$

Hence, if a quaternion is a unit quaternion, then

$$q^{-1} = q^* \quad (6.15)$$

6.3.2. Rotation Through Quaternions

let w be a unit vector describing the axis of rotation, and θ the angle of rotation. The unit quaternion describing the rotation is then represented by

$$q = iw_1 \sin \frac{\theta}{2} + jw_2 \sin \frac{\theta}{2} + kw_3 \sin \frac{\theta}{2} + \cos \frac{\theta}{2} \quad (6.16)$$

A rotation of a vector x about the angle and rotation vector can then be written as

$$y = qxq^{-1} \quad (6.17)$$

As the rotation can always be represented by a rotation matrix, let $y = Rx$, which yields the following relation:

$$Rx = qxq^{-1} \quad (6.18)$$

We can then find the rotation matrix represented with quaternions. Solving for R to find the rotation matrix described by the elements of q yields [20]:

$$\mathbf{R} = \begin{pmatrix} 1 - 2(q_2^2 + q_3^2) & 2(q_1q_2 - q_3q_4) & 2(q_1q_3 + q_2q_4) \\ 2(q_1q_2 + q_3q_4) & 1 - 2(q_1^2 + q_3^2) & 2(q_2q_3 - q_1q_4) \\ 2(q_1q_3 - q_2q_4) & 2(q_2q_3 + q_1q_4) & 1 - 2(q_1^2 + q_2^2) \end{pmatrix} \quad (6.19)$$

Quaternion for rotations is preferred over Euler Angles due to easier computations and the alleviation of the gimbal lock problems. However, quaternions still suffer from other sources of trouble. Quaternions suffer from ambiguity caused by their anti-podal symmetry: q and $-q$ correspond to the same rotation, hence quaternions double cover the $SO(3)$ group. Nevertheless, this can be solved by restricting the scalar part to be positive.

The mapping in Eq. 6.19 is shown to satisfy all the desirable constraints but pre-images connectivity and the notion of continuity [54, 6]. The pre-image of any rotation matrix represented by a non-zero quaternion q consists in $\{\alpha q | \alpha \in \mathbb{R}^*\}$, which is not connected [6].

6.4. Gram-Schmidt Orthogonalization

6.4.1. 5D and 6D representations of rotations

Another mapping suggested in [54] is the Gram-Schmidt orthogonalization procedure. The Gram-Schmidt procedure is as follows:

Let V be a inner product space, and $x, v \in V, v \neq 0$. The orthogonal projection of the vector x onto the vector v is:

$$p = \frac{\langle x, v \rangle}{\langle v, v \rangle} v \quad (6.20)$$

The remainder $o = x - p$ is orthogonal to v . This process can be extended to arbitrarily number of vectors: if v_1, v_2, \dots, v_n is an orthogonal set of vectors

$$p = \frac{\langle x, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1 + \frac{\langle x, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2 + \dots + \frac{\langle x, v_n \rangle}{\langle v_n, v_n \rangle} v_n \quad (6.21)$$

In which p is the orthogonal projection of the vector x onto the subspace spanned by v_1, \dots, v_n , i.e., the remainder $o = x - p$ is orthogonal to v_1, \dots, v_n .

This serves as the basis for the Gram Schmidt orthogonalization process. Let V be a vector space with an inner product. Suppose x_1, x_2, \dots, x_n is a basis for V . Then the Gram Schmidt orthogonalization process is as follows:

$$\begin{aligned}
v_1 &= x_1, & e_1 &= \frac{v_1}{\|v_1\|} \\
v_2 &= x_2 - \frac{\langle x_2, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1, & e_2 &= \frac{v_2}{\|v_2\|} \\
v_3 &= x_3 - \frac{\langle x_3, v_1 \rangle}{\langle v_1, v_1 \rangle} v_1 - \frac{\langle x_3, v_2 \rangle}{\langle v_2, v_2 \rangle} v_2, & e_3 &= \frac{v_3}{\|v_3\|} \\
&\vdots & & \vdots \\
v_n &= x_n - \sum_{i=1}^{n-1} \frac{\langle x_n, v_i \rangle}{\langle v_i, v_i \rangle} v_i, & e_n &= \frac{v_n}{\|v_n\|}
\end{aligned}$$

where v_1, v_2, \dots, v_k is an orthogonal basis for V , whilst e_1, \dots, e_n is an orthonormal. Zhou et al. proposes to use the Gram Schmidt procedure for a 3×2 matrix (6D) as a mapping function. Zhou et al. further shows that the dimensionality can be reduced to 5D with a stereo-graphic projection, quoting worse performance than 6D, but better than the traditional methods. The mapping does however satisfy the desirable properties set forth in [6]

6.5. Symmetric Orthogonalization via SVD

Consider a rotation matrix $\mathbb{R} \in SO(3)$ with a singular value decomposition as $R = U\Sigma V^T$. Then the symmetric orthogonalization with SVD is:

$$SVDO(M) = UV^T \tag{6.22}$$

whereas the special orthogonalization:

$$SVDO^+(M) = U\Sigma'V^T \tag{6.23}$$

where $\Sigma' = \text{diag}(1, \dots, 1, \det(UV^T))$ maps to $SO(3)$.

Orthogonalization via SVD consistently outperform the other mapping functions in [6, 24] and satisfies all the desirable properties in [6]. The mapping was proposed by [24], with an extensive analysis over its properties. This section will go through the origin of the mapping and properties in a greater detail than the previous mappings, as this is the current state-of-the-art method.

The reasoning for using the singular value decomposition comes from the Procrustes problem which is a sub-problem of Wahba's problem. The goal was to estimate the position of a space craft based on the observations of the position of

the stars relative to the space craft.

6.5.1. Wahba's Problem

Wahba's problem was originally posed by Grace Wahba in 1965 [49] and typically involves finding an optimal rotation to fit a series of vector measurements. The origin of the problem stems from attitude estimation in the aerospace field. How can the orientation of a space-craft be determined by making remote observations of celestial bodies of reference points? Given two sets of m n -dimensional points $\{v_1, v_2, \dots, v_m\}$ and $\{v_1^*, v_2^*, \dots, v_m^*\}$, find a rotation matrix $R \in SO(3)$ such that the loss function L is minimized.

$$L(\mathbf{R}) = \frac{1}{2} \sum_{i=1}^n a_i \|v_i - \mathbf{R}v_i^*\|^2 \quad (6.24)$$

Figure 6.1 illustrates the problem with $m = 4, n = 3$ where v_i are representations of the four observed objects from the representation frames, whereas v_i^* are observed from the space body. a_i are weights associated with every observation. One of the proposed solutions was derived by [32], which includes the usage of singular value decomposition.

Assume that the weights are normalized, i.e., $\sum_{i=1}^n a_i = 1$, then:

$$L(\mathbf{R}) = 1 - \sum_{i=1}^n a_i v_i^T \mathbf{R} v_i^* = 1 - \text{tr}(\mathbf{R}\mathbf{A}) \quad (6.25)$$

where:

$$\mathbf{A} = \sum_{i=1}^n v_i (v_i^*)^T \quad (6.26)$$

Find the singular value decomposition of the matrix \mathbf{A} :

$$\mathbf{A} = \mathbf{U}\mathbf{\Sigma}\mathbf{A}^T \quad (6.27)$$

The rotation matrix can then be found by:

$$\mathbf{R} = \mathbf{A}\mathbf{\Sigma}'\mathbf{A}^T \quad (6.28)$$

where $\mathbf{\Sigma}' = \text{diag}([1, 1, \det(\mathbf{A}), \det(\mathbf{A})])$

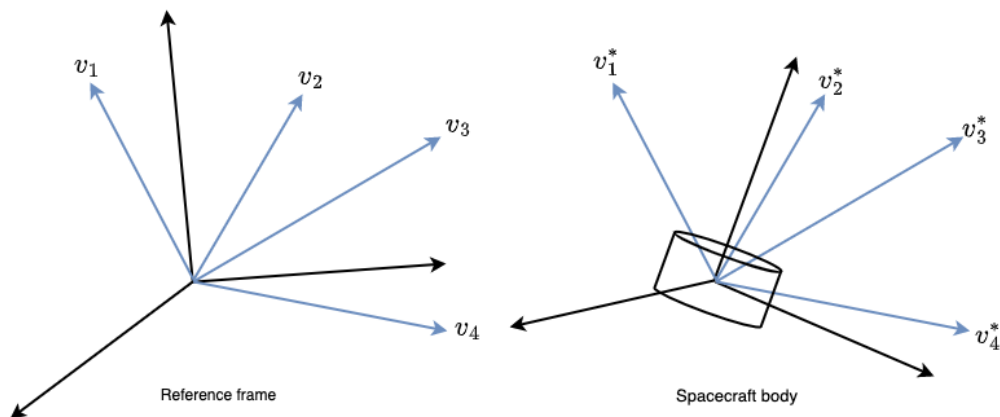


Figure 6.1.: The known reference frame is on the left and space craft's reference frame is on the right. Wahba's problem is to find a rotation matrix such that the orientation of the space craft can be known. Figure adopted from [28]

6.5.2. Procrustes Problem

A special case of Wahba's problem is the orthogonal Procrustes problem. Consider two matrices \mathbf{A} and \mathbf{A} , then the problem is to find an orthogonal matrix $\mathbf{\Omega}$ which closely maps \mathbf{A} to \mathbf{A} :

$$\mathbf{R} = \arg \min_{\mathbf{\Omega} \in O(3)} \|\mathbf{\Omega} \mathbf{A} - \mathbf{A}\|_F \quad (6.29)$$

Where $\|\cdot\|_F$ denotes the Frobenius norm. The solution was first solved by Peter Schonemann [43]. The problem is equivalent to

$$\min_{\mathbf{R} \in O(3)} \|\mathbf{R} - \mathbf{M}\|_F \quad (6.30)$$

where $\mathbf{M} = \mathbf{A} \mathbf{A}^T$. Then to find the matrix \mathbf{R} , the singular value decomposition is taken of \mathbf{M} :

$$\mathbf{M} = \mathbf{A} \mathbf{\Sigma} \mathbf{A}^T \quad (6.31)$$

and then

$$\mathbf{R} = \mathbf{A} \mathbf{A}^T \quad (6.32)$$

The proof is as follows:

$$\begin{aligned}
\mathbf{R} &= \arg \min_{\mathbf{\Omega} \in O(3)} \|\mathbf{\Omega A} - \mathbf{B}\|_F^2 \\
&= \arg \min_{\mathbf{\Omega} \in O(3)} \langle \mathbf{\Omega A} - \mathbf{B}, \mathbf{\Omega A} - \mathbf{B} \rangle_F \\
&= \arg \min_{\mathbf{\Omega} \in O(3)} \|\mathbf{\Omega A}\|_F^2 + \|\mathbf{B}\|_F^2 - 2\langle \mathbf{\Omega A}, \mathbf{B} \rangle_F \\
&= \arg \min_{\mathbf{\Omega} \in O(3)} \|\mathbf{A}\|_F^2 + \|\mathbf{B}\|_F^2 - 2\langle \mathbf{\Omega A}, \mathbf{B} \rangle_F \\
&= \arg \max_{\mathbf{\Omega} \in O(3)} \langle \mathbf{\Omega}, \mathbf{B A}^T \rangle_F \\
&= \arg \max_{\mathbf{\Omega} \in O(3)} \langle \mathbf{\Omega}, \mathbf{U \Sigma V}^T \rangle_F \\
&= \arg \max_{\mathbf{\Omega} \in O(3)} \langle \mathbf{U}^T \mathbf{\Omega V}, \mathbf{\Sigma} \rangle_F \\
&= \arg \max_{\mathbf{\Omega} \in O(3)} \langle \mathbf{S}, \mathbf{\Sigma} \rangle_F
\end{aligned} \tag{6.33}$$

As \mathbf{S} is an product of orthogonal matrices, \mathbf{S} is also orthogonal, and hence the expression is maximized when \mathbf{S} equal the identity matrix \mathbf{I}^1 , thus:

$$\mathbf{I} = \mathbf{U}^T \mathbf{R V} \tag{6.34}$$

and hence

$$\mathbf{R} = \mathbf{U V}^T \tag{6.35}$$

Note that in Eq. 6.33, to go from step 3 to 3, the property of orthogonality was used for $\mathbf{\Omega}$:

$$\|\mathbf{\Omega A}\|_F^2 = \text{tr}((\mathbf{\Omega A})^T (\mathbf{\Omega A})) = \text{tr}(\mathbf{A}^T \mathbf{\Omega}^T \mathbf{\Omega A}) = \text{tr}(\mathbf{A}^T \mathbf{A}) = \|\mathbf{A}\|_F^2 \tag{6.36}$$

In order to ensure special orthogonality, the problem can be solver with

$$\mathbf{R} = \mathbf{U \Sigma' V}^T \tag{6.37}$$

Where $\mathbf{\Sigma}'$ is $\text{diag}([1, 1, \det(\mathbf{U V}^T)])$ which is equal to $SVDO^+$, as in section 6.5. For this reason, symmetric orthogonalization via SVD is sometimes referred to as a Procrustes mapping. The solution proves that symmetric orthogonalization is optimal in the least-square sense.

¹Note that for orthogonal matrices, individual entries cannot have a larger absolute value than 1.

6.5.3. Properties of Symmetric Srthogonalization

Special orthogonalization via SVD can be shown to be a maximum likelihood estimate which minimizes the expected error in the presence of Gaussian Noise [24]. Let $\mathbf{M} = \mathbf{R}_\mu + \Sigma \mathbf{N}$, $\mathbf{M} \in \mathbb{R}^{n \times n}$ be an observation of $\mathbf{R}_\mu \in SO(n)$, corrupted by noise \mathbf{N} with entries $n_{ij} \sim \mathcal{N}(0, 1)$. The probability density function of the matrix normal distribution of the likelihood function is [15]:

$$L(\mathbf{R}_\mu; \mathbf{M}, \Sigma) = \left((2\pi)^{\frac{n^2}{2}} \Sigma^{n^2} \right)^{-1} \exp \left(-\frac{1}{2\Sigma^2} \left((\mathbf{M} - \mathbf{R}_\mu)^T (\mathbf{M} - \mathbf{R}_\mu) \right) \right) \quad (6.38)$$

The likelihood $L(\mathbf{R}_\mu; \mathbf{M}, \Sigma)$ where $\mathbf{R}_\mu \in SO(n)$ is maximized when the term $\left((\mathbf{M} - \mathbf{R}_\mu)^T (\mathbf{M} - \mathbf{R}_\mu) \right)$ is minimized:

$$\arg \max_{\mathbf{R}_\mu \in SO(n)} L(\mathbf{R}_\mu; \mathbf{M}; \Sigma) = \arg \min_{\mathbf{R}_\mu \in SO(n)} (\mathbf{M} - \mathbf{R}_\mu)^T (\mathbf{M} - \mathbf{R}_\mu) = \arg \min_{\mathbf{R}_\mu \in SO(n)} \|\mathbf{M} - \mathbf{R}_\mu\|_F^2 \quad (6.39)$$

As shown in 6.5.2, the solution is given by $SVDO^+(\mathbf{M})$, due to the fact that it is optimal in the least-squares sense. Hence the $SVDO^+(\mathbf{M})$ is a maximum likelihood estimate.

6.5.4. Gradients

This section is based on the findings of [24, 10].

The stability and continuity of the gradients is an important factor for stability during training of an network. Recall the backpropagation calculation for an SVD layer in 4.

Let \mathbf{P} denote a singular value decomposition:

$$\mathbf{M} = \mathbf{U}\Sigma\mathbf{V}^T \quad (6.40)$$

Consider the closest orthogonal matrix:

$$\mathbf{R} = \mathbf{U}\mathbf{V}^T \quad (6.41)$$

Set the loss function L as the Frobenius norm squared:

$$L = \left\| \mathbf{U}\mathbf{V}^T - \mathbf{P} \right\|_F^2 \quad (6.42)$$

This can be rewritten as (see 3.1.7):

$$L = \text{tr} \left(\left(\mathbf{UV}^T - \mathbf{P} \right) \left(\mathbf{UV}^T - \mathbf{P} \right) \right) = \text{tr}(\mathbf{I} + \mathbf{PP}^T - 2\mathbf{UV}^T\mathbf{P}^T) \quad (6.43)$$

And

$$\text{tr}(2\mathbf{UV}^T\mathbf{P}^T) = \text{tr}(\mathbf{U}(\mathbf{PV})^T) = \text{tr}(\mathbf{U}^T(\mathbf{PV})) = \text{tr}(\mathbf{VU}^T\mathbf{P}) \quad (6.44)$$

such that:

$$\frac{\partial L}{\partial \mathbf{U}} = -2 \frac{\partial \text{tr}(\mathbf{UV}^T\mathbf{P}^T)}{\partial \mathbf{U}} = -2\mathbf{PV} \quad (6.45)$$

$$\frac{\partial L}{\partial \mathbf{U}} = -2 \frac{\partial \text{tr}(\mathbf{UV}^T\mathbf{P}^T)}{\partial \mathbf{U}} = -2\mathbf{P}^T\mathbf{U} \quad (6.46)$$

To find the gradients, recall Eq. 4.47. The expressions above are the partial gradients, and we can begin to insert into the expression in Eq. 4.47. Denote the terms within the square brackets for \mathbf{U} and \mathbf{V} as:

$$\mathbf{C} = \mathbf{U}^T \frac{\partial \mathbf{f}}{\partial \mathbf{U}} - \frac{\partial \mathbf{f}^T}{\partial \mathbf{U}} = -2(\mathbf{U}^T\mathbf{P}^T\mathbf{V} - \mathbf{V}^T\mathbf{P}\mathbf{U}) \quad (6.47)$$

and

$$\mathbf{D} = \mathbf{V}^T \frac{\partial \mathbf{f}}{\partial \mathbf{V}} - \frac{\partial \mathbf{f}^T}{\partial \mathbf{V}} = -2(\mathbf{V}^T\mathbf{P}^T\mathbf{U} - \mathbf{U}^T\mathbf{P}\mathbf{V}) \quad (6.48)$$

Notice that $\mathbf{C} = -\mathbf{D}$, and insert into Eq. 4.47:

$$dL = \mathbf{U} \left(\mathbf{F}^{-1} \circ \mathbf{C}\mathbf{\Sigma} \right) \mathbf{V}^T + \mathbf{U} \left(\mathbf{I}_k \frac{\partial L}{\partial \mathbf{\Sigma}} \right) \mathbf{V}^T + \mathbf{U} (\mathbf{\Sigma}\mathbf{F}^{-1} \circ -\mathbf{C}) \mathbf{V}^T \quad (6.49)$$

Simplifying ($\frac{\partial L}{\partial \mathbf{\Sigma}} = 0$):

$$dL = \mathbf{U} (\mathbf{F}^{-1} \circ \mathbf{C}\mathbf{\Sigma} - \mathbf{\Sigma}\mathbf{F}^{-1} \circ) \mathbf{V}^T \quad (6.50)$$

By inspecting the terms $\mathbf{F}^{-1} \circ \mathbf{C}\mathbf{\Sigma}$ and $\mathbf{\Sigma}(\mathbf{F}^{-1} \circ \mathbf{C})$

$$\left\{ \mathbf{F}^{-1} \circ \mathbf{C}\mathbf{\Sigma} \right\}_{ij} = \frac{\mathbf{C}_{ij}\sigma_j}{\sigma_i^2 - \sigma_j^2}, \quad \left\{ \mathbf{\Sigma}(\mathbf{F}^{-1} \circ \mathbf{C}) \right\}_{ij} = \frac{\mathbf{C}_{ij}\sigma_i}{\sigma_i^2 - \sigma_j^2} \quad (6.51)$$

Define $\mathbf{Z} = (\mathbf{F} \circ \mathbf{C})\boldsymbol{\Sigma} - \boldsymbol{\Sigma}(\mathbf{F} \circ \mathbf{C})$ and

$$\frac{\partial L}{\partial \mathbf{M}} = \mathbf{U}\mathbf{Z}\mathbf{V}^T \quad (6.52)$$

where

$$\mathbf{Z}_{ij} = \begin{cases} \frac{-\mathbf{C}_{ij}}{\sigma_i + \sigma_j}, & \text{if } i \neq j \\ 0, & \text{if } i = j \end{cases} \quad (6.53)$$

From Eq. 6.53 it is clear that the gradient for $SVDO(M)$ is undefined if two singular values are zero, or very large when a sum of two singular values is near zero. Extending the analysis to $SVDO^+$ is the same if $\det(M) > 0$. For the other case of $\det M < 0$ there is an extra factor of $D = \text{diag}(1, 1, \dots, -1)$, which changes the last singular values sign. The denominator in Eq. 6.53 is either $\sigma_i - \sigma_n$ or $\sigma_n - \sigma_j$ if either of i or j is equal to n . This causes the gradient to be undefined if the smallest singular value is equal to another singular value. The gradient is also large if another singular value is near the smallest.

The calculations above shows that both $SVDO$ and $SVDO^+$ are continuous and differentiable, except for the discontinuity if the determinant $\det(M) < 0$ and the smallest singular value has a multiplicity of maximum 1. These conditions are rare for small matrices such as a 3×3 rotation matrix.

6.5.5. Comparison with Gram-Schmidt

The Gram-Schmidt orthogonalization in 6D can be seen as a degenerate case of $SVDO^+$, expressed as the limit:

$$GS_{6D}(M) = \lim_{\alpha \rightarrow 0^+} \arg \min_{\mathbf{R} \in SO(3)} \left\| \mathbf{R} \begin{pmatrix} 1 & 0 \\ 0 & \alpha \\ 0 & 0 \end{pmatrix} - \mathbf{M} \right\|_F^2 \quad (6.54)$$

The proof is illustrated in [6]. Solving this optimization problem for a given α is equivalent to solving the Procrustes problem, considering $\mathbf{M} \begin{pmatrix} 1 & 0 & 0 \\ 0 & \alpha & 0 \end{pmatrix} \in \mathbb{R}^{3 \times 3}$ as the input. This shows that both mappings share some properties, except that the 6D Gram-Schmidt mapping almost exclusively gives importance to the first column of \mathbf{M} , and is only well defined when its rank is 2. For further details about the connection, see [6].

6.6. Summary

Table 6.1 summarizes the different mapping functions discussed and how they relate to the desirable properties set forth by Brégier [6], which recommends $SVDO^+$ as the mapping function.

	Domain	Surjective & differentiable	Full rank Jacobian	Connected/convex pre-images
Euler Angles	\mathbb{R}^3	✓	×	×
Quaternion	\mathbb{R}^3	✓	✓	×
GS-6D	$\mathbb{R}^{3 \times 2}$	✓	✓	✓
$SVDO^+$	$\mathbb{R}^{3 \times 3}$	✓	✓	✓

Table 6.1.: Table showcasing which of the desirable properties each mapping function satisfies. Adopted from [6]

Chapter 7.

Experiments

The following chapter includes a showcase and explanation of the data sets and pre-processing, before explaining the experiments. Four experiments were conducted, each related to rotations pose estimation in 3D, in addition to an extra pose refinement test. The goal of the experiments is to find out the potential of special orthogonalization with SVD as a rotation representation in deep learning.

All the models were written in PyTorch [38], which includes an off-the-shelf solver for singular value decomposition. PyTorch also has the feature of “AutoGrad“ which automatically calculates the gradients in the backward pass for any operation induces in the forward pass. This eases the troubles of comparing the rotation representations, as the back-propagation is not required to implement manually. All the models were trained on NTNUs high performance cluster [44], which allowed for multi-GPU training. The training procedure is time-consuming and hence only the first experiment includes all the rotation representations. After, the main focus is on the performance of symmetric orthogonalization $SVDO^+$. The code can be viewed on [github](#).

7.1. Datasets

7.1.1. ModelNet

ModelNet10 [51] is a data set containing 4899 instances of 3D CAD models from 10 different categories. The different models are seperated into a train set 80% and a test set 20%. As the samples come in CAD models, they need to be rendered. The usual goal of the data set is to for classification, but in this case it is used for pose estimation regression w.r.t the reference point (i.e., camera). The 3D CAD objects be turned into point clouds, or rendered into 2D images.

The classes for ModelNet10 are everyday household items such as a toilet, sofa,

desk, bed et cetera. One object from each category is shown in Figure 7.1 with a rotation matrix:

$$R = \begin{pmatrix} 0.5 & 0 & \frac{\sqrt{3}}{2} \\ 0 & 1 & 0 \\ -\frac{\sqrt{3}}{2} & 0 & 0.5 \end{pmatrix} \quad (7.1)$$

The 3D CAD objects can also be used for creating point clouds. A point cloud is a collection of nodes sampled from the models. Each point cloud is on the form $\mathbb{R}^{N \times 3}$ where N is the number of nodes, and 3 is from each nodes position x, y, z . Four samples of point clouds are illustrated in Figure 7.2.

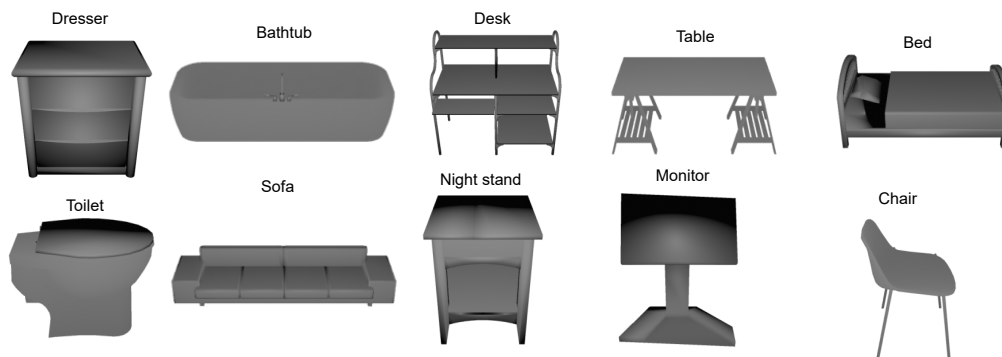


Figure 7.1.: All ten classes from ModelNet10 [51], rendered by me.

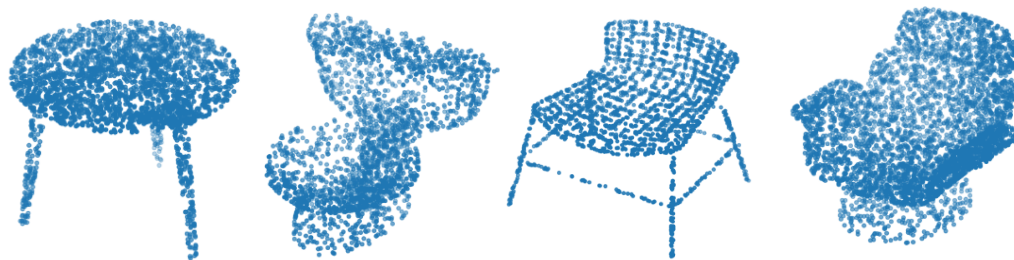


Figure 7.2.: Example of point clouds from ModelNet10 with 2500 sampled nodes.

As rendering is a task with a lot of different parameters (such as the positioning, intensity, and cone shape of lighting, the size of the objects et cetera), Liao, Gavves, and Snoek created ModelNet10-SO(3) [27], for an ease of comparison between different models. This was used for 3D pose estimation regression to ensure a fair comparison. The pre-rendered data set includes the class label and annota-

tions for the rotation matrix. The data set was created by rendering 100 images from every object per class in the train set, and 4 from the test set.

7.1.2. UPNA

The UPNA head pose data set [4] is a data set for head tracking and position estimation. The data set consist of 120 videos of 10 subjects (6 men and 4 females), with synchronized labels for key points, rotations, and translations. Figure 7.3 shows four samples from the data set:



Figure 7.3.: Four samples from the UPNA head pose data set [4]

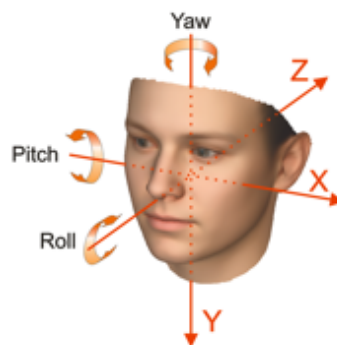


Figure 7.4.: The rotation axes for a human head. Figure taken from [4]

7.2. Comparison test

A comparison between the mappings was conducted by generating random rotation matrices and transforming a coordinate system. The transformed system was used as input, and the goal was to regress the rotation matrix. This can be viewed as an auto-encoder, as the initial coordination system is the identity matrix. This means that the transformed coordinates is the same as the rotation matrix.

Three variants of the experiment was conducted, where the difference was the maximum allowed rotation around each axis. The limit was 45, 90 and 180 degrees. For each experiment 30 000 random rotations were generated and a batch size of 64 was used. The models were trained until the learning stagnated for all networks in case some of the representations took longer to learn. The learning rate was initially set to 0.01, but was halved for every tenth epoch beginning starting from epoch 20.

The neural network used for this task was a simple feed forward network with two

hidden layers. The input is a coordination system in 3D, i.e., a 3×3 matrix, which is flattened into 1×9 . The first hidden layer has a size of 128 neurons, whereas the second has only 64. The output layer dimension varies according to the given rotation representation. The output is then parameterized to $SO(3)$ before the loss function is applied for back-propagation. The Frobenius norm was used for all models as the loss function. The network architecture is illustrated in Figure 7.5.

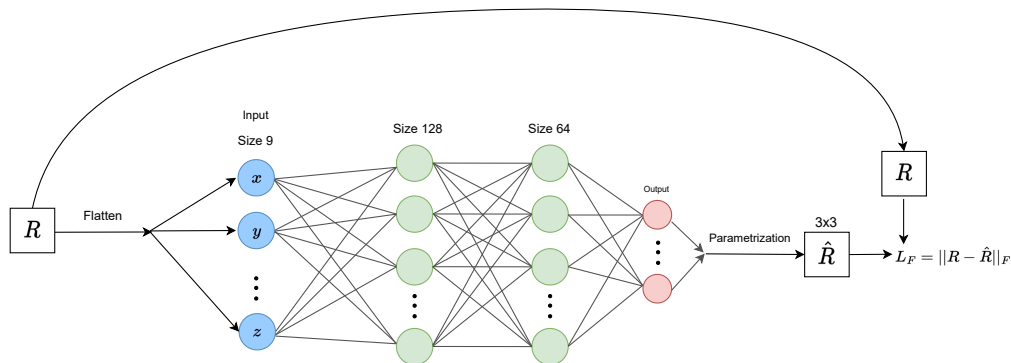


Figure 7.5.: The network architecture for the preliminary experiment. Notice how the output layer varies in dimensions due to the different parametrization methods.

The following rotation representations with the network output f dimension are as follows:

1. Euler angles: $f \in \mathbb{R}^3$
2. Quaternions: $f \in \mathbb{R}^4$
3. Gram Schmidt orthogonalization in 5D: $f \in \mathbb{R}^5$
4. Gram Schmidt orthogonalization in 6D: $f \in \mathbb{R}^6$
5. Symmetric orthogonalization via SVD: $f \in \mathbb{R}^9$
6. Direct regression without orthogonalization: $f \in \mathbb{R}^9$. This is a true rotation representation, as there is no guarantee it will be orthogonal. It is included nonetheless to see how well the network is able to perform without a parametrization function. The output is simply reshaped into 3×3 .

All the models use stochastic gradient descent with an initial learning rate of 0.01, which is halved every tenth epoch beginning from the 20th. The evaluation metric used is the mean angle error between the output rotation matrix and the

target. Note that for the direct regression, there are errors. Recall the angle error formula:

$$\theta = \arccos \frac{\text{tr } R - 1}{2} \quad (7.2)$$

As the trace of R is not guaranteed to be smaller than 2 due to the orthogonality constraint breached. For this reason, the value inside the arccos was clamped to $[-1, 1]$ for the direct regression, which is not a correct procedure. As will be shown, this has no true effect on the outcome.

7.3. 3D Head Pose Estimation from 2D images of humans

7.3.1. Setup & Pre-Processing

The pre-processing procedure is the same as in [36]. From the key point labels, a bounding box for the face is created for each image, effectively capturing just the face. Afterwards, a small random perturbation of the given bounding box is performed. This allows one to augment the data set, as the original data set is quite limited. Figure 7.6 shows one sample from the data set cropped to the size 224×224 with the rotation matrix:



$$R = \begin{pmatrix} 0.9346 & 0.3536 & 0.0378 \\ 0.3449 & -0.9272 & 0.1459 \\ 0.0866 & -0.1233 & -0.9886 \end{pmatrix}$$

Figure 7.6.: One sample from the UPNA data set cropped to 224×224 .

7.4. 3D Pose Estimation from 2D Images

The 3D pose estimation task is to predict the pose of an object compared to a standard reference pose from a single 2D image. Depending on the max allowed rotation, this is seen as a difficult problem. The data set used is the ModelNet10-SO(3) [27], which allows all rotations in $\mathbf{R} \in SO(3)$. Earlier attempts to solve this task on the ModelNet10-SO(3) data set has been made by [27, 36, 24]. [27] uses quaternions as the rotation representation and discretizes the angle error output into bins rather than directly regressing. These bins are normalized and used as a probability distribution. Mohlin, Bianchi, and Sullivan attempts to overcome this issue by using a neural network to output the parameters for a matrix Fisher distribution. Levinson et al. trains a standard ResNet-50 [17] for the same rotation representations presented in this paper, but omits the classes exhibiting rotational symmetry, whilst these are included for this experiment.

As the ResNet-RS has significant improvements over the out-dated ResNet, the experiment is redone in this paper. A ResNet-RS101 is employed for the task and both symmetric orthogonalization via SVD and Gram Schmidt 6D are tested as a rotation representation. The network is trained on all ten classes, and tested per class afterwards for the median and the mean. In addition the accuracy of the mean angle error beneath three thresholds of 30, 15, and 7.5 degrees is quoted.

7.4.1. Network Architecture

The network architecture used for the 3D regression task was ResNet-RS101 [5]. The network achieved a top-1 accuracy of 84 % on ImageNet, making it state of the art in image recognition. Even though this experiment is to regress for rotation matrices, and not for classification purposes, pretrained weights were used. This is to utilize the potential of transfer learning, as the already trained ResNet-RS already has managed to learn and extract a variety of filter maps. The pre-trained weights works as a boot-strap for the training process and significantly decreases the training time. The last layer of the network is modified into a feed forward network with the size of 2048 and outputs the dimension suitable for the rotation representation.

The general architecture with symmetric orthogonalization as mapping is shown in Figure 7.7. A rotated object is rendered and used as an image which is fed through the 101 layers of ResNet-RS. The output is flattened and used as input to a feed forward network which reduces the dimensionality suitable for the rotation representation. Then the orthogonalization process takes place, until the loss is finally calculated with the Frobenius norm.

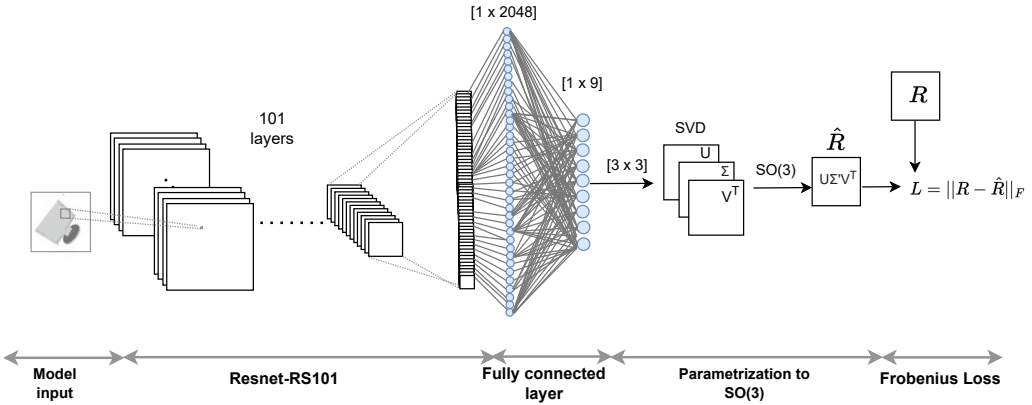


Figure 7.7.: A general overview of a typical network architecture with orthogonalization step. This figure shows symmetric orthogonalization via SVD.

7.5. 3D Iterative Pose alignment

The last and final experiment is an extra to test whether the symmetrical orthogonalization with SVD has the ability to reduce the estimated errors even further.

Whereas the 3D object pose estimation is a harder problem, the 3D iterative pose alignment problem is able to improve the initial guesses made by the other network.

The process can be viewed as a refinement process for a pose estimation process. Consider two images: the ground truth pose I_{gt} and an initial guess I_i . The goal is to find the transformation matrix R that maps the object to the camera by estimating the relative rotation and translation between the initial guess and the ground truth.

The network architecture is inspired by DeepIM [25] and the CosyPose Model [22]. There are differences however. Both DeepIM and CosyPose are models for pose estimation in 6D, i.e., both rotations and translations. Further, DeepIM uses quaternions as the rotation parametrization and CosyPose utilizes the 6D Gram-Schmidt orthogonalization.

This experiment is a simple to showcase the potency of the model architectures. A neural network takes both the ground truth and the initial guess input, including a depth map for the ground truth. A depth map is simply an image with higher intensity the closer the object is. The input is sent through the network with and outputs a vector of size \mathbb{R}^9 . These are then parametrized to $SO(3)$. The new rotation matrix estimate is $R^{init} R^i$. The rotation matrix is combined with a fixed

translation vector, and used to create the transformation matrix. This is then used in turn to render a new image of the object, and the procedure is repeated. This is done k times, or until a certain threshold is reached.

The Frobenius loss was replaced with the loss function ADDL1-loss. The reason is that ADDL1-loss uses the randomly sampled nodes from the 3D CAD object. These nodes are then transformed them with the initial guess and the ground truth, and the distance between each node is calculated and averaged as loss.

$$\text{ADD} - 1 = \frac{1}{H_l} \sum_h \left\| TX_l^h - \hat{T} X_l^h \right\|_2 \quad (7.3)$$

\hat{T} is the estimated transformation matrix, and T is the ground truth. H_l is the number of sampled nodes for object l , which was set to 750 for every object in this experiment.

As rendering is a time-consuming and CPU-heavy process, only a maximum of two iterations per epoch was used. The model was only trained on the chair class for the ModelNet10 data set. The model was first trained on on the initial guesses for 100 epochs, then another the number of iterations were turned up to 2 for another 100 epochs.

7.5.1. Network Architecture

The network architecture is the same as for 3D pose estimation with a few modifications. Instead of using the ResNet-RS101, the original ResNet101 was used instead. The reason is that the input consists of seven channels, making it hard to utilize the pretrained weights of ResNet-RS. There was however a modification implemented in PyTorch making it possible to adapt the original ResNet101 with pretrained weights for a chosen number of channels[37].

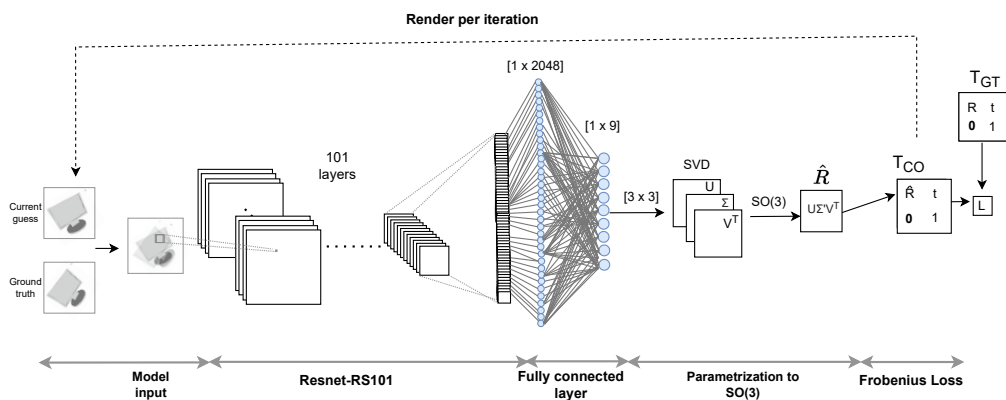


Figure 7.8.: The network architecture for the last experiment. The initial guess is used to render an 2D image, which is combined with the ground truth image. The combination, as well as a depth map, is used as input for the neural network. Parametrization with SVD is used to map to $SO(3)$. The rotation matrix is combined with a fixed translation, and another image is rendered and the procedure begins from start, with the rendered image as initial guess.

Chapter 8.

Results and discussion

The following chapter showcases the results for all the experiments. Each experiment is shown in a separate section, with the following discussion.

8.1. Comparison Test for Mapping Functions

The comparison test gave insights into how the different rotation representation mappings perform. Table 8.1 shows the mean angle error for every rotation representation after 50 epochs for a maximum rotation of 45, 90, and 180 degrees for the test set. There are significant differences in training loss and the outcome of the test set; both are showcased.

Max Rot	45	90	180
Direct	>100	>100	>100
Euler	0.67	3.3	23.33
Quaternion	0.39	0.62	2.42
GS-5D	0.59	0.85	1.11
GS-6D	0.43	0.63	0.91
SVD	0.38	0.53	0.72

Table 8.1.: Mean angle errors for the preliminary experiment for every rotation representation for max rotations of 45, 90 and 180 degrees. Symmetric orthogonalization via SVD consistently outperforms for any maximum rotation. Further, notice that the performance of quaternions is second best for maximum rotation as 45 and 90 degrees, and the true potency of Gram-Schmidt is not exhibited before the rotations are a maximum of 180 degrees.

Figures 8.1, 8.3, 8.5 shows the mean angle error in degrees for all epochs during the training. To differentiate between the models at the end of the training, the Figures 8.2, 8.4 and 8.6 shows the last 25 epochs with a maximum angle error of

0.8 degrees. Direct regression has an extremely poor performance, which makes the need for a rotation representation evident.

Further it is clear that the Euler angles are consistently out-performed, especially for rotations above 45 degrees. What is surprising is that the quaternions actually outperforms the 5D Gram-Schmidt orthogonalization representations for every rotation during the training. However, the results for the test set in Table 8.1, it is the opposite. This might indicate some over-fitting for the quaternions. For the test set, the trend is as expected. Euler performs the worst, quaternions second, then the Gram-Schmidt orthogonalization in 5D and 6D respectively, and finally symmetric orthogonalization consistently outperforms all the other representations.

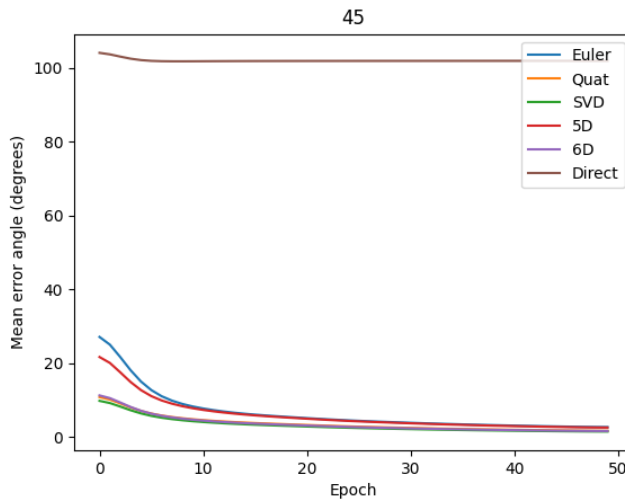


Figure 8.1.: The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 45 per axis. Notice how direct regression is extremely poor, hence yielding solid evidence for the need of a rotation representation layer.

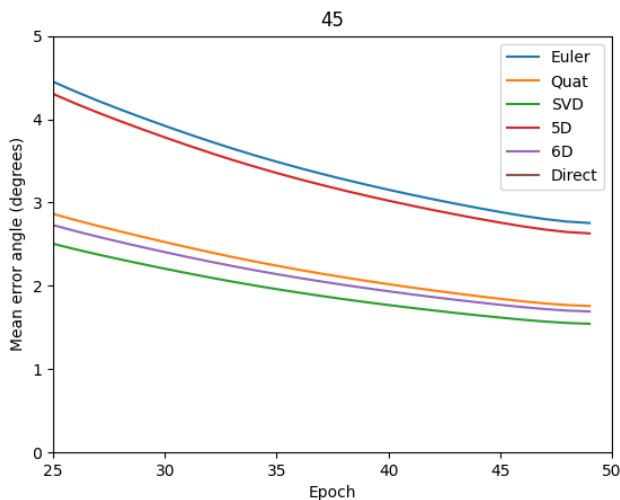


Figure 8.2.: The mean angle error between the true rotation matrix restricted to 0-4 degrees from epoch 25-50 with rotations up to 45 per axis. It is clear that the SVD has the best performance, and Euler angles the worst.

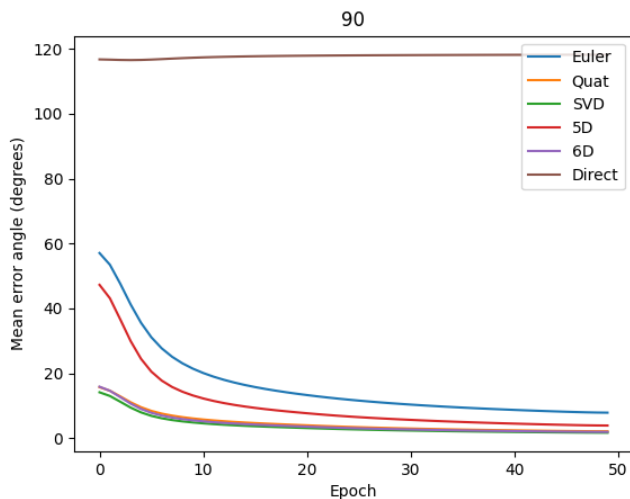


Figure 8.3.: The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 45 per axis. The trend of superiority of SVD continues, whereas Euler angles still has the worst performance.

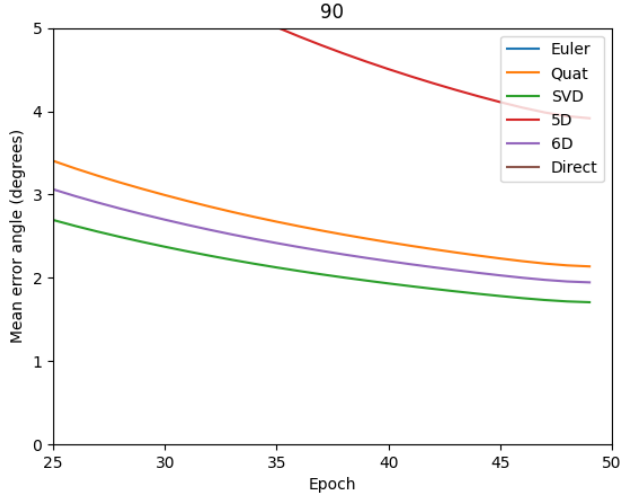


Figure 8.4.: The mean angle error between the true rotation matrix restricted to 0-4 degrees from epoch 25-50 with rotations up to 90 per axis.

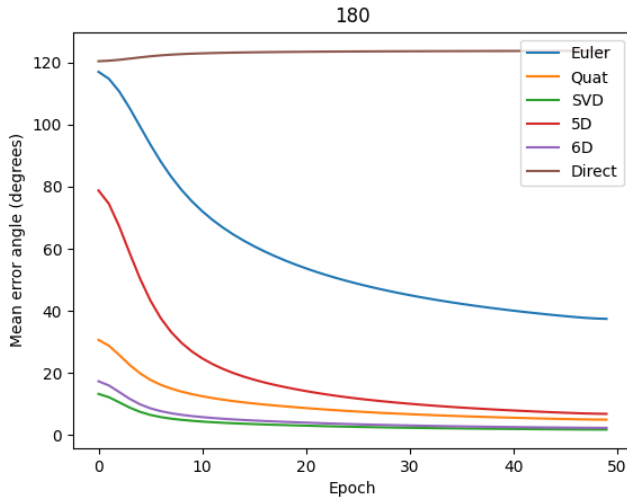


Figure 8.5.: The mean angle error between the true rotation matrix R and the estimated \hat{R} with rotations up to 180 per axis. SVD still has the best performance, whilst Euler angles stagnates with a mean angle error of 40 degrees. This highlights how bad the Euler angles are as a representation for rotations, as it cannot even learn to map the income to itself.

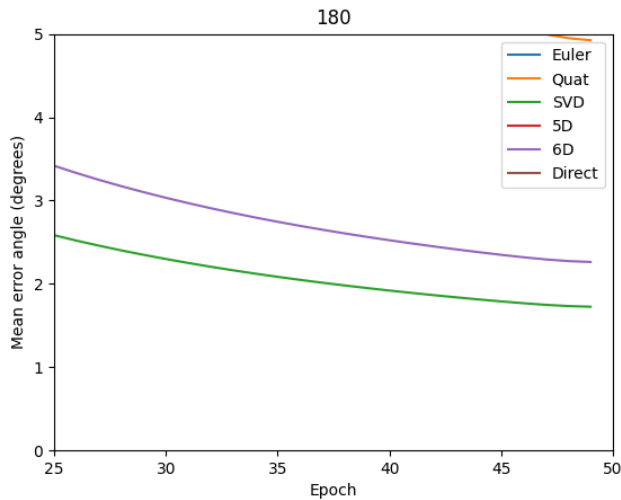


Figure 8.6.: The mean angle error between the true rotation matrix restricted to 0-4 degrees from epoch 25-50 with rotations up to 180 per axis. Only the SVD and GS-6D models was able to yield mean angle errors below 5 degrees for rotations up to 180 degrees. This showcases that regressing for full rotations around every axis is a difficult problem.

8.2. UPNA Head Pose

For the UPNA head pose estimation, the best obtained mean angle error was 7.49 degrees. The authors of the data set was only able to obtain an mean angle error of 8.3, with a different test set. This test set included subjects used both in the training and test set, meaning that the the test set employed in this experiment is harder. Nevertheless, [36] is able to obtain a performance of 4.3 degrees, by regressing to a Fisher matrix distribution.

The reason for the poor performance is likely due to the extreme learning abilities for ResNet-RS101. The test angle was reduced to below 3 degrees within 5 epochs. Due to the small amount of data, the model overfit. To alleviate this problem, weigh decay was added in addition to a drop-out rate of 0.5. The Result was an improvement to an angle error of 5.7, still lower than for [36]. It is likely that with more optimization and hyper-parameter adjustments, the performance would have improved more. Figure 8.7 shows the train and test loss for the two main runs with $SVDO^+$.

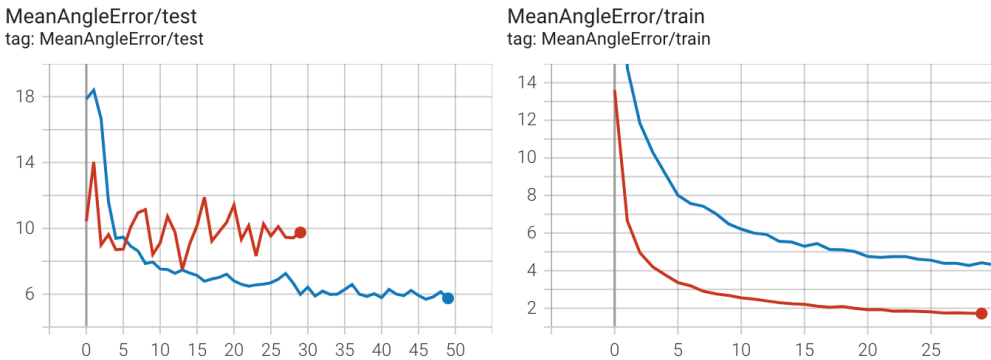


Figure 8.7.: The results from two of the runs on the UPNA data set. The red line is the model without any penalization for overfitting, whereas the blue line is the model with weight decay and drop-out rate of 0.5. Nevertheless, the angle error never fell below 5.7 degrees.

8.3. 3D object pose estimation from 2D images

The results from the 3D pose estimation experiments is highlighted in Table 8.3. Comparisons are made against [36] and [27]¹. The metrics used are mean and median angle error as well as accuracy within a certain threshold. Only the main findings are showcased in this section, with a more detailed showcase in Appendix B.

	Mean Error	Median Error	Acc $\frac{\pi}{6}$	Acc $\frac{\pi}{12}$	Acc $\frac{\pi}{24}$
VGG16+S $_{exp}^3$ [27]	-	20.3	70.9	58.9	38.4
VGG16+S $_{exp}^3$ revised	-	28.7	65.8	49.6	35.2
Fisher [36]	-	17.1	75.7	69.3	55.2
GS-6D Adjusted	19.93	8.61	84.14	71.7	42.6
<i>SVDO</i> ⁺ Adjusted	18.47	7.53	85.2	73.9	49.8

Table 8.2.: Results for all ten classes in ModelNet10-SO(3). Adjusted refers to symmetry adjustment.

8.3.1. Symmetry in the samples

Some of the classes has objects which have rotational symmetry, such as the table class. If a table is rotated 180 degrees around, it is exactly the same. This is a problem in 3D pose estimation, as the rotation matrices does have the same values, except some of the signs in the rotation matrix are opposites. Hence if this is not adjusted for, the metrics for evaluating performance will highly biased. Figure 8.8 shows the distribution of the angle errors before adjusting for the rotational symmetry for the table class. This was adjusted for by selecting the angles above 150 degrees, then comparing the mean angle error if the objects was rotated 180 degrees around one of the axis. The reasoning is that if the objects are identical with a 180 degree rotation it is irrelevant whether or the item is rotated around the symmetrical axis in the real world. However, the rotation matrix will have opposite signs in some of the entries, causing the model to be penalized due a high loss value, when in fact, the estimation was correct.

¹The original paper had an error in the calculation for the angle error. This is the adjusted and true result of [27]. See [26] for further information.

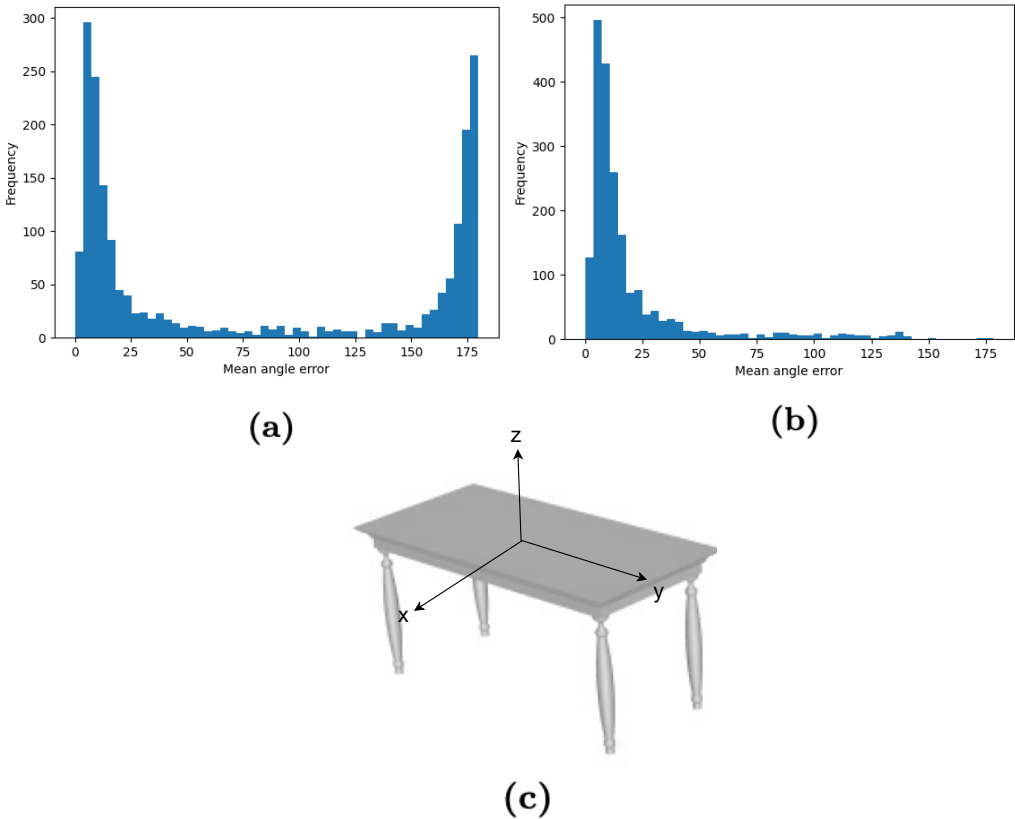


Figure 8.8.: Distribution of angle errors before and after adjusting for symmetry. Notice that in a) there are two modes, in each corner of the histogram. The mean is 93.33 degrees, furthering the belief that the model cannot distinguish between 180 degrees or 0 degrees, as expected. To account for this, the rotations were rotated by 180 for the symmetrical axes when calculating the mean error. The resulting distribution is shown in b). Figure c) shows how a table has rotational symmetry when rotated around the z axis 180 degrees. This procedure significantly lowered the mean angle error, as the new mean in b) is 19.83.

Figure 8.9 shows some of the estimated poses in comparison to the true pose. For the majority of the batches, the results are very good. However, there are still some results that are way off. Appendix B includes 100 overlays from the predicted pose and the ground truth pose from samples in the test set. Some outliers has an error of up to 96 degrees, which makes it difficult to use in an industrial settings. However, as will be shown in the next sections, a refinement process may be applied to alleviate this problem.

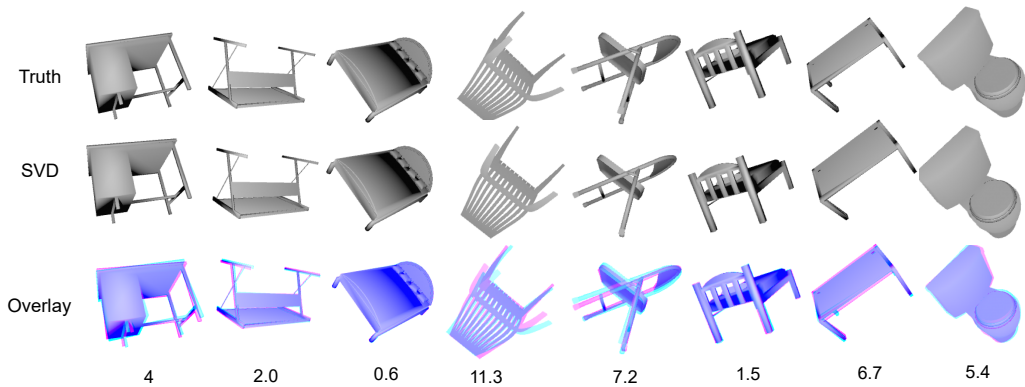


Figure 8.9.: The upper row shows some of the rotations in the test set of ModelNet10-S0(3). The lower row shows the estimated rotations with the use of symmetric orthogonalization via SVD.

	Sofa			Chair			Toilet			bed		
	Mn ↓	Md ↓	σ ↓	Mn ↓	Md ↓	σ ↓	Mn ↓	Md ↓	σ ↓	Mn ↓	Md ↓	σ ↓
[24]	18.01	7.31	33.96	21.25	11.14	30.28	-	-	-	-	-	-
Adjusted GS-6D	10.57	5.88	17.01	14.	8.18	20.26	9.51	6.64	12.69	11.86	6.7	18.78
Adjusted <i>SVDO</i> ⁺	15.3	5.13	35.47	15.26	6.75	28.73	9.39	5.08	21.18	23.28	5.41	48.88

Table 8.3.: Selected classes from Modelnet10 3D pose estimation results in comparison with [24], who only report results for the Sofa and Chair class. Mn, Md, and σ are abbreviations for mean, median, and standard deviation.

	Table			Bathtub		
	Mn ↓	Md ↓	σ ↓	Mn ↓	Md ↓	σ ↓
Original	93.33	97.22	73.85	79.9	34.23	75.95
Adjusted GS-6D	18.75	8.7	28.18	28.43	15.156	35.49
Adjusted <i>SVDO</i> ⁺	19.83	10.35	26.65	28.29	14.4	32.52

Table 8.4.: Example of classes with a high presence of symmetrical objects around one or more axes. Notice how the mean of the error angle of table class (every table in the data set is symmetrical around the y-axis.) is 90 degrees. The error reduces to only 19.83 when this symmetry is accounted for.

8.4. Iterative 3D Pose Refinement

Figure 8.10 shows the results on some of the models from the test set. Column one shows the ground truth, whereas column two shows the model input (the images overlay, but the depth map is removed for visibility). The last two columns are the first and second model guess respectively. The annotations above each overlay shows the angle error between the poses.

The Table 8.5 shows the mean and median for the initial rotation, and the first and second model pose estimate for the test set. The mean for the initial guess is 15, which is not surprising as the rotations were uniformly samples with a max degree of 30. Further, it is clear that the first guess of the model is not able to improve the estimate. However, the second guess is able to improve the estimate by 33% for the mean angle error, and 50% for the median.

	Initial guess	First model guess	Second model guess
Mean Angle Error	15.05	15.07	10.09
Median Angle Error	16.99	17.04	7.94

Table 8.5.: The mean angle error for the model. The initial guess is around 15 degrees as expected, as the rotations varied random uniformly for a maximum rotation of 30 degrees.

The model was only tested on a limited training set; hence it is very likely that the model’s performance may be improved by more optimization. For the epochs north of 50, the training loss quickly approached zero. The poor generalization ability is likely due to overfitting the data. Augmenting the data set and adding more samples will likely impact the mean error loss significantly. Further, increasing the number of iterations per epoch and utilizing the refinement process is probable to improve the models.


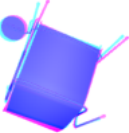

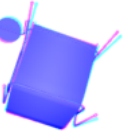





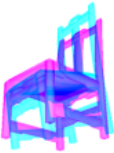
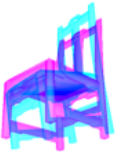








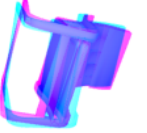
GT	Initial input	First guess	Second Guess
	5.5 	5.7 	1.83 
	24.8 	24.9 	9.15 
	16.8 	16.9 	4.23 
	25 	24.4 	17.3 
	27.5 	27.4 	7.7 

Figure 8.10.: The refinement process for randomly selected models from the test set. The numbers represent the angle error between the rotations. The initial guess in column three is more often equal to the initial guess, even slightly worse. However, there is a significant decrease in the error from columns three to four, indicating that the refinement process is working.

Chapter 9.

Conclusion

The experiments strengthened the theoretical findings regarding the different rotation representations and the potency of using singular value decomposition as a mapping between the Euclidean space and the rotation space. The comparison test confirmed that direct regression for rotation regression is not an option. It further confirmed the assumption made in 6 that the Euler angles are unfit for rotation representations due to the discontinuities and Jacobian rank deficit due to Gimbal lock. Lastly, it showcases the superiority of the higher dimensional methods such as Gram Schmidt 6D and the SVD orthogonalization as a mapping function.

Moreover, a performance surpassing the comparative works was reached by utilizing $SVDO^+$ with state-of-the-art CNN architecture such as ResNet-RS101. We see that the SVD orthogonalization as a mapping function yields significant improvements over the traditional 3D image pose estimation with traditional mapping in [36, 27]. However, none of [36, 27, 54, 24] discusses how to deal with rotational symmetry; instead, the works ignore the classes exhibiting it. For classes with symmetry, this was adjusted by rotating the ground truth by 180 degrees for objects with rotational symmetry, such as the table class. The lack of symmetry adjustment will lead to significant upwards bias in the estimation mean and median angle error. For further work, the adjustment procedure should be included during the training, removing the penalty for correct rotations in the real world but incorrect rotation matrices. Taking this into account during the training might increase the performance further.

Further, the last experiment illuminates the possibility of training an iterative network to increase the estimate with a few iterations for a small error in an initial guess. A natural step for further work would be to connect the two trained networks for end-to-end pose estimation from a single 2D image and its 3D CAD model.

As always with machine learning, the results are susceptible to hyper-parameters. Testing other optimization algorithms (e.g., Adam, RMSprop, et cetera), batch sizes, or learning rates will likely enhance the results further. Higher resolution images, or other network architectures can be deployed or train for more epochs with increased data.

References

- [1] Arthur Alaniz and Christina Marianne G. Mantaring. “Real-Time Camera Pose Estimation for Virtual Reality Navigation”. In: 2010.
- [2] Stephen Andrilli and David Hecker. “Chapter 1 - Vectors and Matrices”. In: *Elementary Linear Algebra (Fifth Edition)*. Ed. by Stephen Andrilli and David Hecker. Fifth Edition. Boston: Academic Press, 2016, pp. 1–83. ISBN: 978-0-12-800853-9. DOI: <https://doi.org/10.1016/B978-0-12-800853-9.00001-3>. URL: <https://www.sciencedirect.com/science/article/pii/B9780128008539000013>.
- [3] G.B. Arfken and H.J. Weber. *Mathematical Methods For Physicists International Student Edition*. Elsevier Science, 2005. ISBN: 9780080470696. URL: <https://books.google.no/books?id=tNtijk2iBSMC>.
- [4] Mikel Ariz, José J. Bengoechea, Arantxa Villanueva, and Rafael Cabeza. “A novel 2D/3D database with automatic face annotation for head tracking and pose estimation”. In: *Computer Vision and Image Understanding* 148 (2016). Special issue on Assistive Computer Vision and Robotics - "Assistive Solutions for Mobility, Communication and HMI", pp. 201–210. ISSN: 1077-3142. DOI: <https://doi.org/10.1016/j.cviu.2015.04.009>. URL: <https://www.sciencedirect.com/science/article/pii/S1077314215000934>.
- [5] Irwan Bello, William Fedus, Xianzhi Du, Ekin D. Cubuk, Aravind Srinivas, Tsung-Yi Lin, Jonathon Shlens, and Barret Zoph. *Revisiting ResNets: Improved Training and Scaling Strategies*. 2021. DOI: [10.48550/ARXIV.2103.07579](https://doi.org/10.48550/ARXIV.2103.07579). URL: <https://arxiv.org/abs/2103.07579>.
- [6] Romain Brégier. *Deep Regression on Manifolds: A 3D Rotation Case Study*. 2021. DOI: [10.48550/ARXIV.2103.16317](https://doi.org/10.48550/ARXIV.2103.16317). URL: <https://arxiv.org/abs/2103.16317>.
- [7] Mai Bui, Tolga Birdal, Haowen Deng, Shadi Albarqouni, Leonidas Guibas, Slobodan Ilic, and Nassir Navab. *6D Camera Relocalization in Ambiguous Scenes via Continuous Multimodal Inference*. 2020. DOI: [10.48550/ARXIV.2004.04807](https://doi.org/10.48550/ARXIV.2004.04807). URL: <https://arxiv.org/abs/2004.04807>.

- [8] Hui Cheng and K. C. Gupta. “An Historical Note on Finite Rotations”. In: *Journal of Applied Mechanics* 56.1 (Mar. 1989), pp. 139–145. ISSN: 0021-8936. DOI: [10.1115/1.3176034](https://doi.org/10.1115/1.3176034). eprint: https://asmedigitalcollection.asme.org/appliedmechanics/article-pdf/56/1/139/5460585/139_1.pdf. URL: <https://doi.org/10.1115/1.3176034>.
- [9] Christopher Clapham and James Nicholson. *Fundamental Theorem of Algebra*. 2009. DOI: [10.1093/acref/9780199235940.013.1201](https://www.oxfordreference.com/view/10.1093/acref/9780199235940.013.1201). URL: <https://www.oxfordreference.com/view/10.1093/acref/9780199235940.001.0001/acref-9780199235940-e-1201>.
- [10] Olav Egeland. “*Optimization*”. unpublished. 2022.
- [11] L. Euler. “Problema algebraicum ob affectiones prorsus singulares memorabile. *Commentatio 407 indicis Enestrœmiani, Novi commentarii academiae scientiarum Petropolitanae*”. it. In: *L.Euleri Opera Omnia, 1st series, Vol. 6* (1770), pp. 287–315.
- [12] Zan Gojcic, Caifa Zhou, Jan D. Wegner, Leonidas J. Guibas, and Tolga Birdal. *Learning multiview 3D point cloud registration*. 2020. DOI: [10.48550/ARXIV.2001.05119](https://arxiv.org/abs/2001.05119). URL: <https://arxiv.org/abs/2001.05119>.
- [13] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. <http://www.deeplearningbook.org>. Cambridge, MA, USA: MIT Press, 2016.
- [14] Henrik Grüner. “*Regularization in machine learning based on mirror descent*”. unpublished. 2021.
- [15] Arjun Kumar Gupta and D. K Nagar. *Matrix variate distributions*. Chapman & Hall/CRC, 2000.
- [16] Richard Hartley and Andrew Zisserman. *Multiple View Geometry in Computer Vision*. 2nd ed. New York, NY, USA: Cambridge University Press, 2003. ISBN: 0521540518.
- [17] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. *Deep Residual Learning for Image Recognition*. 2015. DOI: [10.48550/ARXIV.1512.03385](https://arxiv.org/abs/1512.03385). URL: <https://arxiv.org/abs/1512.03385>.
- [18] Suzana Herculano-Houzel. “The human brain in numbers: a linearly scaled-up primate brain”. In: *Frontiers in human neuroscience* 3 (2009), p. 31.
- [19] Catalin Ionescu, Orestis Vantzos, and Cristian Sminchisescu. “Training deep networks with structured layers by matrix backpropagation”. In: *arXiv preprint arXiv:1509.07838* (2015).
- [20] Mehdi Jafari and Yusuf Yayli. “Generalized quaternions and rotation in 3-space $\mathbb{3}$ ”. In: *space* 4 (2012), p. 2.

- [21] Abhijit Kundu, Yin Li, and James M. Rehg. “3D-RCNN: Instance-Level 3D Object Reconstruction via Render-and-Compare”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2018), pp. 3559–3568.
- [22] Yann Labbé, Justin Carpentier, Mathieu Aubry, and Josef Sivic. *Cosy-Pose: Consistent multi-view multi-object 6D pose estimation*. 2020. DOI: [10.48550/ARXIV.2008.08465](https://doi.org/10.48550/ARXIV.2008.08465). URL: <https://arxiv.org/abs/2008.08465>.
- [23] Yann LeCun, Corinna Cortes, and CJ Burges. “MNIST handwritten digit database”. In: *ATT Labs [Online]*. Available: <http://yann.lecun.com/exdb/mnist> 2 (2010).
- [24] Jake Levinson, Carlos Esteves, Kefan Chen, Noah Snavely, Angjoo Kanazawa, Afshin Rostamizadeh, and Ameesh Makadia. *An Analysis of SVD for Deep Rotation Estimation*. 2020. DOI: [10.48550/ARXIV.2006.14616](https://doi.org/10.48550/ARXIV.2006.14616). URL: <https://arxiv.org/abs/2006.14616>.
- [25] Yi Li, Gu Wang, Xiangyang Ji, Yu Xiang, and Dieter Fox. “DeepIM: Deep Iterative Matching for 6D Pose Estimation”. In: *International Journal of Computer Vision* 128.3 (Nov. 2019), pp. 657–678. DOI: [10.1007/s11263-019-01250-9](https://doi.org/10.1007/s11263-019-01250-9). URL: <https://doi.org/10.1007/s11263-019-01250-9>.
- [26] Shuai Liao, Efstratios Gavves, and Cees G. M. Snoek. *Correction for the Mean Angle Error calculation*. 2022. URL: https://github.com/leoshine/Spherical_%20Regression.
- [27] Shuai Liao, Efstratios Gavves, and Cees G. M. Snoek. *Spherical Regression: Learning Viewpoints, Surface Normals and 3D Rotations on n-Spheres*. 2019. DOI: [10.48550/ARXIV.1904.05404](https://doi.org/10.48550/ARXIV.1904.05404). URL: <https://arxiv.org/abs/1904.05404>.
- [28] Bisharah Libbus, Gordon Simons, and Yi-Ching Yao. “Rotating Multiple Sets of Labeled Points to Bring Them Into Close Coincidence: A Generalized Wahba Problem”. In: *The American Mathematical Monthly* 124 (2017), pp. 149–160.
- [29] Kevin M. Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. 1st. USA: Cambridge University Press, 2017. ISBN: 1107156300.
- [30] Jan R. Magnus and Heinz Neudecker. *Matrix Differential Calculus with Applications in Statistics and Econometrics*. Second. John Wiley, 1999. ISBN: 0471986321 9780471986324 047198633X 9780471986331.
- [31] Eric Marchand, Hideaki Uchiyama, and Fabien Spindler. “Pose Estimation for Augmented Reality: A Hands-On Survey”. In: *IEEE Transactions on Visualization and Computer Graphics* 22.12 (2016), pp. 2633–2651. DOI: [10.1109/TVCG.2015.2513408](https://doi.org/10.1109/TVCG.2015.2513408).

- [32] F. Landis Markley. “Attitude determination using vector observations and the singular value decomposition”. In: *Journal of The Astronautical Sciences* 36 (1988), pp. 245–258.
- [33] Warren S McCulloch and Walter Pitts. “A logical calculus of the ideas immanent in nervous activity”. In: *The bulletin of mathematical biophysics* 5.4 (1943), pp. 115–133.
- [34] Xiaoli Meng, Heng Wang, and Bingbing Liu. “A Robust Vehicle Localization Approach Based on GNSS/IMU/DMI/LiDAR Sensor Fusion for Autonomous Vehicles”. In: *Sensors* 17.9 (2017). ISSN: 1424-8220. DOI: [10.3390/s17092140](https://doi.org/10.3390/s17092140). URL: <https://www.mdpi.com/1424-8220/17/9/2140>.
- [35] Thomas P. Minka. *Old and New Matrix Algebra Useful for Statistics*. Tech. rep. 2001.
- [36] D. Mohlin, G. Bianchi, and J. Sullivan. *Probabilistic orientation estimation with matrix Fisher distributions*. 2020. DOI: [10.48550/ARXIV.2006.09740](https://doi.org/10.48550/ARXIV.2006.09740). URL: <https://arxiv.org/abs/2006.09740>.
- [37] Akash Palrecha. *MultiChannel ResNet implementation for PyTorch*. 2022. URL: <https://github.com/akashpalrecha/Resnet-multichannel>.
- [38] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. “PyTorch: An Imperative Style, High-Performance Deep Learning Library”. In: *Advances in Neural Information Processing Systems 32*. Ed. by H. Wallach, H. Larochelle, A. Beygelzimer, F. d’Alché-Buc, E. Fox, and R. Garnett. Curran Associates, Inc., 2019, pp. 8024–8035. URL: <http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>.
- [39] Kaare Brandt Petersen and Michael Syskind Pedersen. *The Matrix Cookbook*. Version 20121115. Nov. 2012. URL: <http://www2.imm.dtu.dk/pubdb/p.php?3274>.
- [40] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. *Searching for Activation Functions*. 2017. arXiv: [1710.05941](https://arxiv.org/abs/1710.05941) [cs.NE].
- [41] O. Rodrigues. “Des lois géométriques qui régissent les déplacements d’un système solide dans l’espace, et de la variation des coordonnées provenant de ces déplacements considérés indépendamment des causes qui peuvent les produire.” fre. In: *Journal de Mathématiques Pures et Appliquées* (1840), pp. 380–440. URL: <http://eudml.org/doc/234443>.

- [42] Ashutosh Saxena, Justin Driemeyer, and A. Ng. “Learning 3-D object orientation from images”. In: *2009 IEEE International Conference on Robotics and Automation* (2009), pp. 794–800.
- [43] Peter Schönemann. “A generalized solution of the orthogonal procrustes problem”. In: *Psychometrika* 31.1 (1966), pp. 1–10. URL: <https://EconPapers.repec.org/RePEc:spr:psycho:v:31:y:1966:i:1:p:1-10>.
- [44] Magnus Själander, Magnus Jahre, Gunnar Tufte, and Nico Reissmann. *EPIC: An Energy-Efficient, High-Performance GPGPU Computing Research Infrastructure*. 2019. arXiv: [1912.05848](https://arxiv.org/abs/1912.05848) [cs.DC].
- [45] Hao Su, Charles R. Qi, Yangyan Li, and Leonidas J. Guibas. “Render for CNN: Viewpoint Estimation in Images Using CNNs Trained with Rendered 3D Model Views”. In: *2015 IEEE International Conference on Computer Vision (ICCV)*. 2015, pp. 2686–2694. DOI: [10.1109/ICCV.2015.308](https://doi.org/10.1109/ICCV.2015.308).
- [46] Jonathan Tremblay, Thang To, Balakumar Sundaralingam, Yu Xiang, Dieter Fox, and Stan Birchfield. *Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects*. 2018. DOI: [10.48550/ARXIV.1809.10790](https://doi.org/10.48550/ARXIV.1809.10790). URL: <https://arxiv.org/abs/1809.10790>.
- [47] Shubham Tulsiani and Jitendra Malik. *Viewpoints and Keypoints*. 2014. DOI: [10.48550/ARXIV.1411.6067](https://doi.org/10.48550/ARXIV.1411.6067). URL: <https://arxiv.org/abs/1411.6067>.
- [48] Charles F Van Loan and G Golub. “Matrix computations (Johns Hopkins studies in mathematical sciences)”. In: *Matrix Computations* (1996).
- [49] Grace Wahba. “A Least Squares Estimate of Satellite Attitude”. In: *SIAM Review* 7.3 (1965), pp. 409–409. DOI: [10.1137/1007077](https://doi.org/10.1137/1007077). eprint: <https://doi.org/10.1137/1007077>. URL: <https://doi.org/10.1137/1007077>.
- [50] Chen Wang, Danfei Xu, Yuke Zhu, Roberto Martín-Martín, Cewu Lu, Li Fei-Fei, and Silvio Savarese. *DenseFusion: 6D Object Pose Estimation by Iterative Dense Fusion*. 2019. DOI: [10.48550/ARXIV.1901.04780](https://doi.org/10.48550/ARXIV.1901.04780). URL: <https://arxiv.org/abs/1901.04780>.
- [51] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. “3D ShapeNets: A deep representation for volumetric shapes”. In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 1912–1920. DOI: [10.1109/CVPR.2015.7298801](https://doi.org/10.1109/CVPR.2015.7298801).
- [52] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. “PoseCNN: A Convolutional Neural Network for 6D Object Pose Estimation in Cluttered Scenes”. In: 2018.
- [53] Han Xiao, Kashif Rasul, and Roland Vollgraf. *Fashion-MNIST: a Novel Image Dataset for Benchmarking Machine Learning Algorithms*. Aug. 28, 2017. arXiv: [cs.LG/1708.07747](https://arxiv.org/abs/cs.LG/1708.07747) [cs.LG].

- [54] Yi Zhou, Connelly Barnes, Jingwan Lu, Jimei Yang, and Hao Li. *On the Continuity of Rotation Representations in Neural Networks*. 2018. DOI: [10.48550/ARXIV.1812.07035](https://doi.org/10.48550/ARXIV.1812.07035). URL: <https://arxiv.org/abs/1812.07035>.

Appendix A.

Supplementary material

A.1. Convolutions

This section comes from the theoretical sections of the preliminary studies [14].

Let $I(i,j)$ be an image, and $K(i,j)$ be the kernel. Then the discrete convolution over the image is denoted as:

$$I(i, j) * K(i, j) = \sum_m \sum_n I(i - m, j - n) K(m, n) \quad (\text{A.1})$$

where $*$ denotes the convolutional operator. A more intuitive explanation is that for each pixel in the input image I , the border-pixels are element-wise multiplied to the kernel. Visually, this can be imagined as the kernel hovers over the pixel in focus, and the elements that match up is multiplied and summed up, resulting in the output pixel. The process for a 5x5 image with 3x3 kernel is shown in Figure [A.1](#).

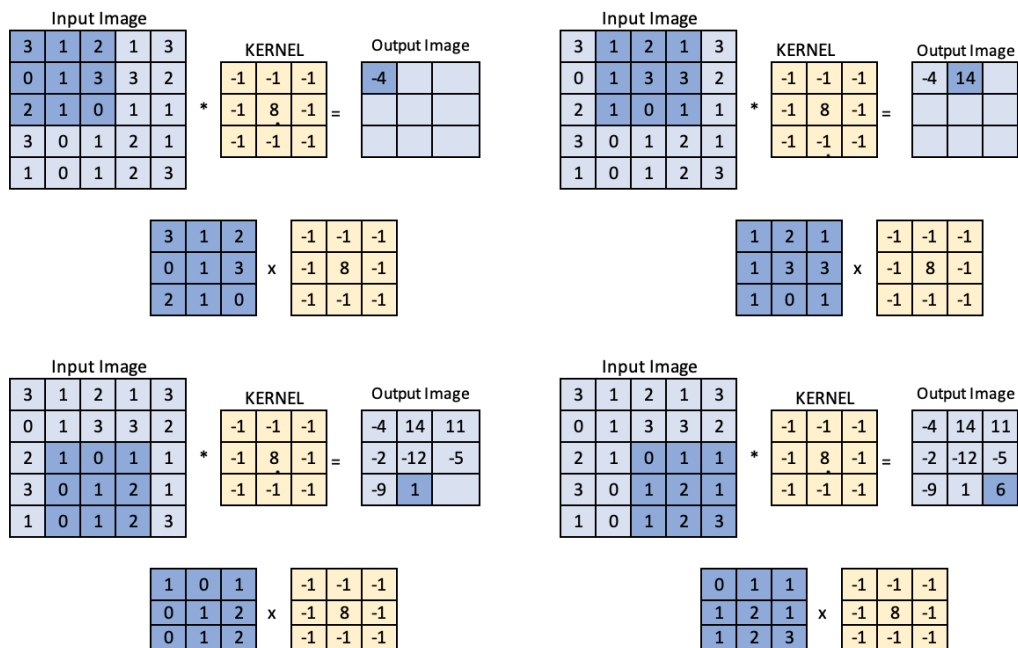


Figure A.1.: Convolution on 5x5 image, with 3x3 kernel size: Upper right is the first convolution. As can be seen, the elements adjacent to the main value is element-wise multiplied with the kernel. The kernel is the Laplacian kernel, which is frequently used for edge detection

Padding

Note that the standard convolution process does not apply the kernel to the borders. To adjust for this, *padding* is often used. This means that artificial cells filled with zeros are put around the border of the image. Figure A.2 shows how the same 3x3 kernel is applied to the 5x5 matrix with padding:

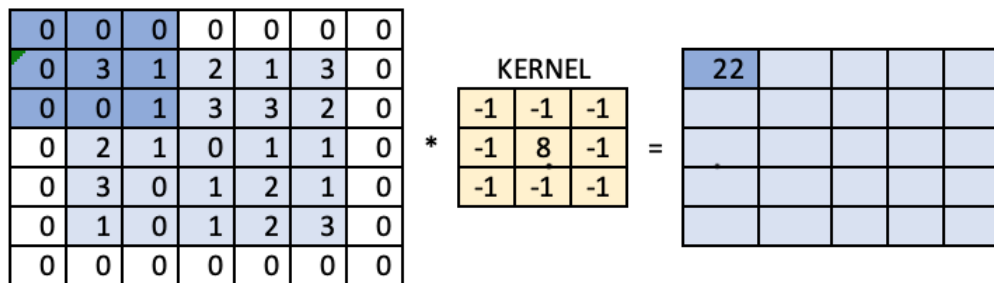


Figure A.2.: First step of a convolution on 5x5 matrix with a 3x3 Laplacian kernel with padding = 1.

Stride

Stride is how far the filter moves then computing the next pixel. In [A.1](#), the stride is 1, because the filter moves to every pixel in the input image. If the stride is set to e.g. 2, the filter will only go to every other pixel. As a consequence, stride > 1 outputs a smaller resolution image than the input.

3	1	2	1	3
0	1	3	3	2
2	1	0	1	1
3	0	1	2	1
1	0	1	2	3

*

KERNEL		
-1	-1	-1
-1	8	-1
-1	-1	-1

=

-13	

3	1	2	1	3
0	1	3	3	2
2	1	0	1	1
3	0	1	2	1
1	0	1	2	3

*

KERNEL		
-1	-1	-1
-1	8	-1
-1	-1	-1

=

-13	17

Figure A.3.: Convolution on 5x5 image with 3x3 Laplacian kernel, zero padding and stride = 2

Appendix B.

3D Pose Estimation - Supplementary Materials

B.1. Distribution of Angle Error

The following figures are the distribution of the angle errors between the true rotation matrix and the estimated, adjusted for errors induced by symmetrical properties of some of the classes. Both the rendered poses and the distributions in the figures shown stems from the $SVDO^+$ model.

	Mean Error	Median Error	Acc $\frac{\pi}{6}$	Acc $\frac{\pi}{12}$	Acc $\frac{\pi}{24}$
Adjusted GS-6D	19.93	8.61	84.14	71.7	42.6
Adjusted $SVDO^+$	18.47	7.53	85.2	73.9	49.8

Table B.1.: Results for 3D pose estimation for all ten classes in ModelNet10-SO(3) adjusted for symmetry

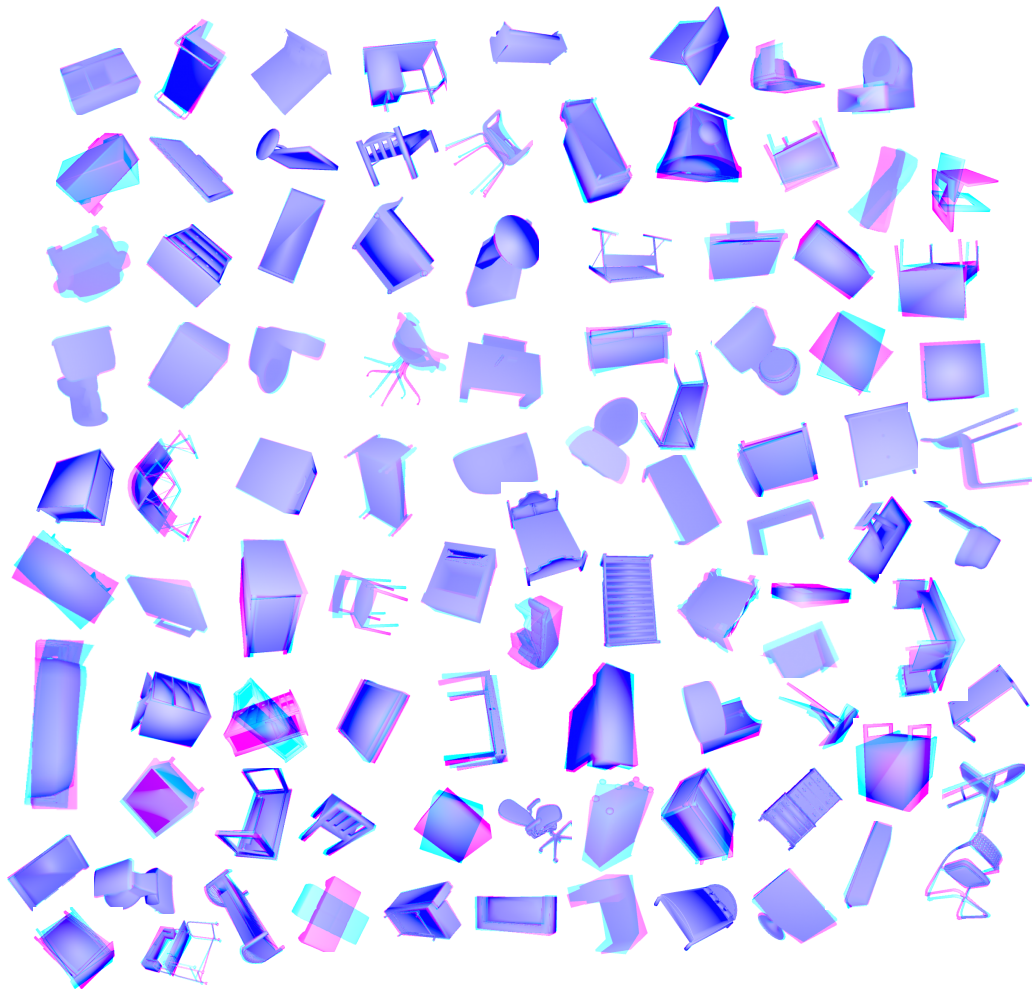
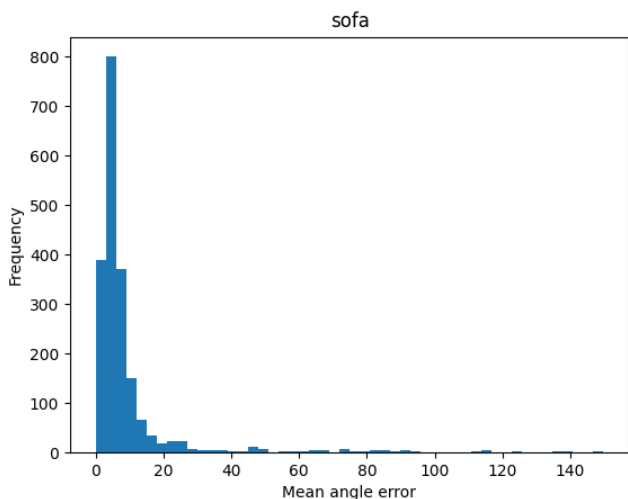


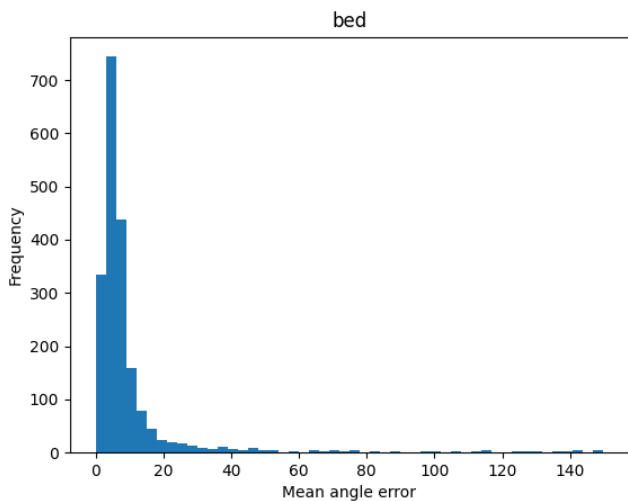
Figure B.1.: Overlay between ground truth pose and a rendered image using the estimated pose from the regression. 100 random samples were taken from the test set with all classes included. The pink is the rendered image, whereas the light blue is the ground truth. If the figure is completely blue, the mean angle error is around 1 degree or less. It is clear that some of the samples drastically increases the mean. The highest angle error for the given test set was 96 degrees, and the lowest were 0.5 degrees.



Sofa	<i>SVDO</i> ⁺	GS-6D
Mean ↓	10.12	10.58
Median	5.21	5.88
Std	18.27	17.02
Acc $\frac{\pi}{6}$ ↑	94.4%	93.95 %
Acc $\frac{\pi}{12}$	88.99%	88.59 %
Acc $\frac{\pi}{24}$	71.14%	63.95 %

Table B.2.: The angle errors for the sofa class

Figure B.2.: The distribution of angle error for sofa class



Bed	<i>SVDO</i> ⁺	GS-6D
Mean ↓	10.79	11.86
Median	5.6	6.69
Std	19.19	18.78
Acc $\frac{\pi}{6}$ ↑	93.75%	93.95 %
Acc $\frac{\pi}{12}$	87.89%	85.59 %
Acc $\frac{\pi}{24}$	67.68%	57.03%

Table B.3.: The angle errors for the bed class

Figure B.3.: The distribution of angle error for bed class

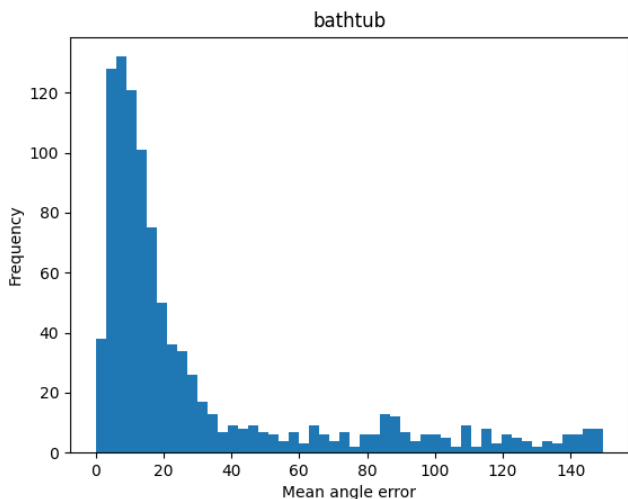


Figure B.4.: The distribution of angle error for bathtub class. Even adjusted for symmetry, the bathtub class was the hardest for the model.

Bathtub	<i>SVDO</i> ⁺	GS
Mn ↓	30.45	28.43
Md	14.44	13.15
Std	36.78	35.49
Acc $\frac{\pi}{6}$ ↑	74%	76.5%
Acc $\frac{\pi}{12}$	52 %	55.6%
Acc $\frac{\pi}{24}$	23.3 %	24.4%

Table B.4.: The distribution of angle errors for the bathtub class

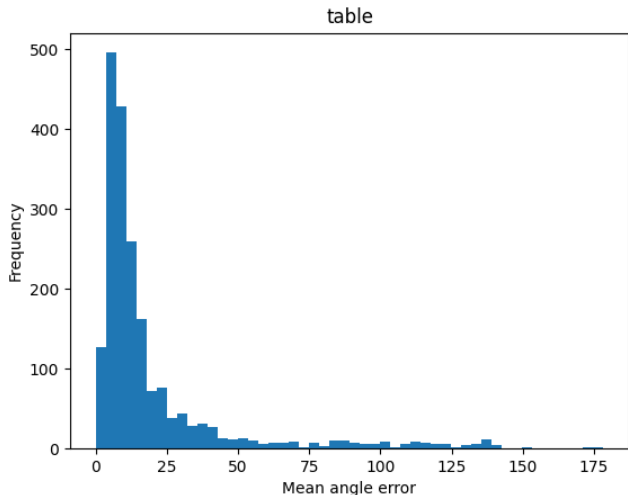
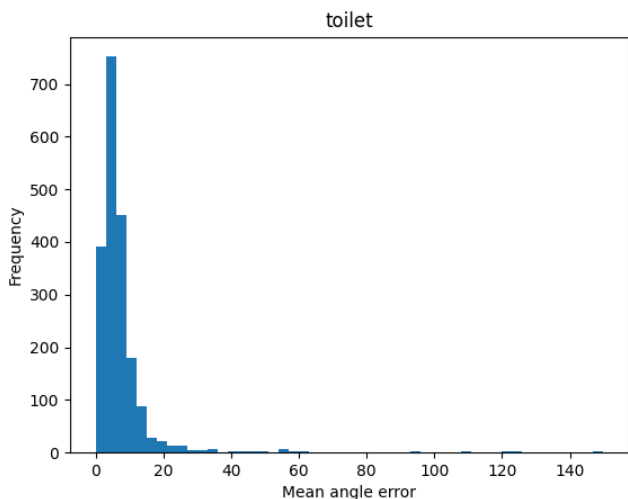


Figure B.5.: The angle errors for the table class

Table	<i>SVDO</i> ⁺	GS-6D
Mean ↓	21.32	18.75
Median	10.36	8.71
Std	30.28	28.19
Acc $\frac{\pi}{6}$ ↑	84.0%	86.74 %
Acc $\frac{\pi}{12}$	67.88%	73.94 %
Acc $\frac{\pi}{24}$	33.72%	41.14 %

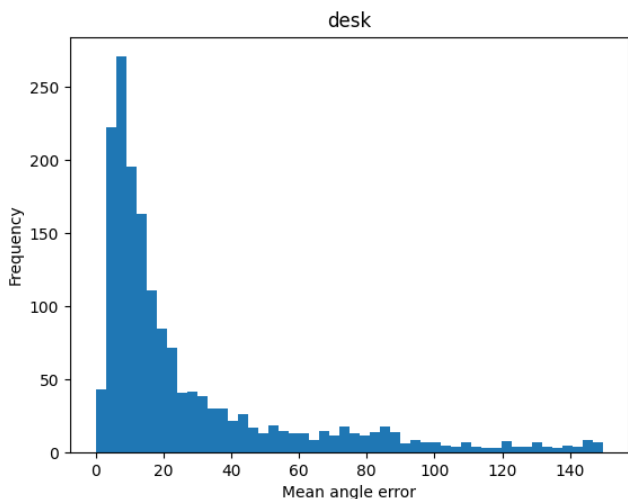
Table B.5.: The distribution of angle errors for the table class



Toilet	<i>SVDO+</i>	GS-6D
Mean ↓	8.11	9.51
Median	5.35	6.64
Std	13.42	12.69
Acc $\frac{\pi}{6}$ ↑	97.3 %	96.8 %
Acc $\frac{\pi}{12}$	93.1 %	89.19 %
Acc $\frac{\pi}{24}$	71.54 %	57.28% %

Table B.6.: The distribution of angle errors for the toilet class

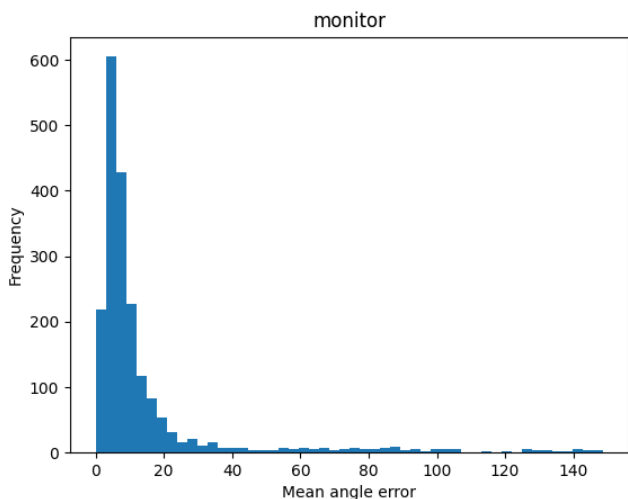
Figure B.6.: The distribution of angle errors for the toilet class



Desk	<i>SVDO+</i>	GS-6D
Mean ↓	27.78	30.28
Median	14.2	14.8
Std	31.71	33.7
Acc $\frac{\pi}{6}$ ↑	72.48 %	70.27 %
Acc $\frac{\pi}{12}$	52.07 %	50.61 %
Acc $\frac{\pi}{24}$	23.33 %	22.28 %

Table B.7.: The angle errors for the desk class

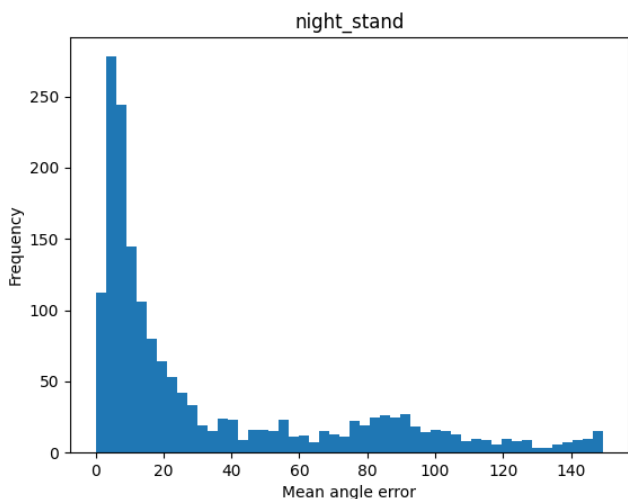
Figure B.7.: The distribution of angle error for desk class



Monitor	<i>SVDO</i> ⁺	GS-6D
Mean ↓	14.67	21.9
Median	6.97	10.46
Std	23.5	30.25
Acc $\frac{\pi}{6}$ ↑	90.15 %	82.94 %
Acc $\frac{\pi}{12}$	79.99 %	65.83 %
Acc $\frac{\pi}{24}$	54.53 %	34.22 %

Table B.8.: The angle errors for the monitor class

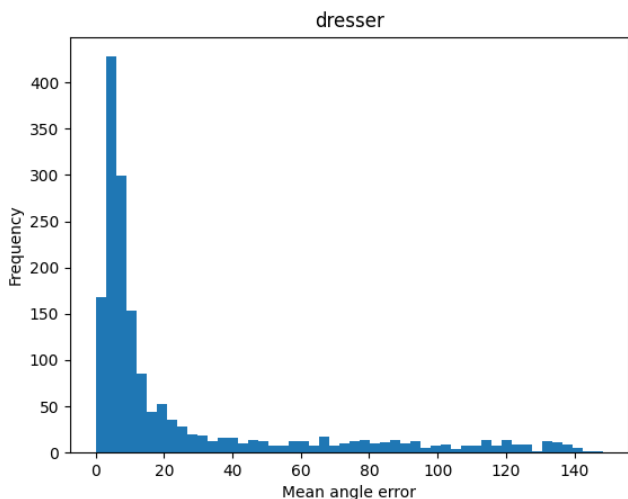
Figure B.8.: The distribution of angle error for monitor class



Night Stand	<i>SVDO</i> ⁺	GS-6D
Mean ↓	33.57	34.2
Median	14.2	13.84
Std	38.5	38.32
Acc $\frac{\pi}{6}$ ↑	67.31 %	66.67 %
Acc $\frac{\pi}{12}$	51.54 %	52.12 %
Acc $\frac{\pi}{24}$	31.18%	29.55 %

Table B.9.: The angle errors for the night stand class

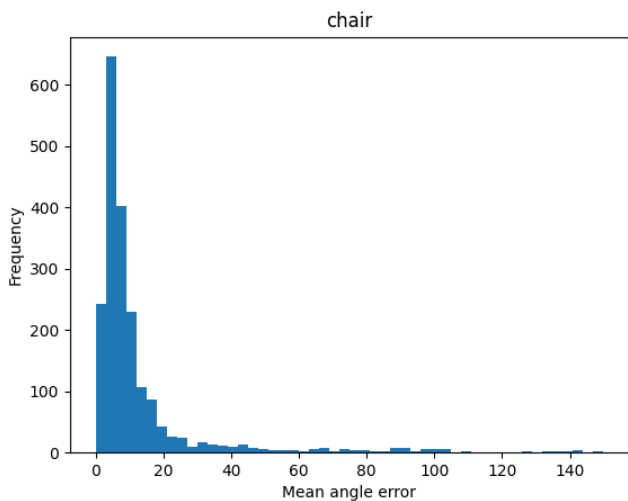
Figure B.9.: The distribution of angle error for night stand class



Dresser	<i>SVDO</i> ⁺	GS-6D
Mean ↓	25.38	28.
Median	8.44	9.92
Std	35.16	36.7
Acc $\frac{\pi}{6}$ ↑	76.61 %	74.52 %
Acc $\frac{\pi}{12}$	66.14 %	62.71 %
Acc $\frac{\pi}{24}$	45.26 %	37.70 %

Table B.10.: The angle errors for the dresser class

Figure B.10.: The distribution of angle error for dresser class



Chair	<i>SVDO</i> ⁺	GS-6D
Mean ↓	13.22	14.63
Median	6.68	8.19
Std	20.87	20.27
Acc $\frac{\pi}{6}$ ↑	90.85 %	89.79%
Acc $\frac{\pi}{12}$	81.49 %	78.29 %
Acc $\frac{\pi}{24}$	56.28 %	44.12 %

Table B.11.: The angle errors for the chair class

Figure B.11.: The distribution of angle error for chair class

