

Augmented Autoencoders for object orientation estimation trained on synthetic RGB images

Martin Sundermeyer



MASTERARBEIT

AUGMENTED AUTOENCODERS FOR OBJECT ORIENTATION ESTIMATION TRAINED ON SYNTHETIC RGB IMAGES

Freigabe:

Der Bearbeiter:

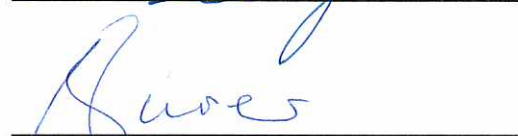
Unterschriften

Martin Sundermeyer



Betreuer:

Maximilian Durner

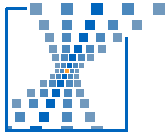


Der Institutsdirektor

Prof. Alin Albu-Schäffer



Dieser Bericht enthält 83 Seiten, 32 Abbildungen und 13 Tabellen



Technische Universität München
Lehrstuhl für Datenverarbeitung
Prof. Dr. Klaus Diepold



Augmented Autoencoders for object orientation estimation trained on synthetic RGB images

Masterarbeit

von

Martin Sundermeyer



Betreuer:

Prof. Dr. Klaus Diepold (TUM)

Johannes Günther (TUM)

Dr. Zoltán-Csaba Márton (DLR)

Maximilian F. P. Durner (DLR)

1. September 2017

Confirmation

Ich, Martin Sundermeyer, versichere hiermit, dass ich die vorliegende Master's Thesis mit dem Titel *Augmented Autoencoders for object orientation estimation trained on synthetic RGB images* selbständig und ohne unzulässige fremde Hilfe erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie wörtliche und sinngemäße Zitate kenntlich gemacht.

Date, Signature

Acknowledgement

This work was carried out in the year 2016/17 at the German Aerospace Center (DLR), at the Institute of Robotics and Mechatronics.

I would like to thank my supervisors at the DLR, Dr. Zoltán-Csaba Márton and Maximilian F.P. Durner for their guidance, support and granted freedom. Furthermore, I want to thank my university supervisors Johannes Günther and Prof. Dr. Klaus Diepold, Head of Chair for Data Processing at the Technical University of Munich (TUM), for the supervision, advice and smooth execution of my thesis.

I am grateful to Prof. Dr.-Ing. Alin Albu-Schäffer, Head of the Institute of Robotics and Mechatronics at the DLR, for giving me the possibility to carry out this work in the department Perception and Cognition.

Additionally, my personal thanks for a lot of useful discussions, collaboration and motivation goes to Dimitri Henkel.

Abstract

Fast and accurate object pose estimation algorithms are crucial for robotic tasks. Despite intensive research, most approaches are not generally applicable on arbitrary object characteristics and dynamic environment conditions.

Learning-based methods like [Convolutional Neural Networks \(CNNs\)](#) have proven good generalization properties given sufficient training data. However, annotating RGB images with 3D object orientations is difficult and requires expert knowledge.

In this work, a real-time approach for joint 2D object detection and 3D orientation estimation is proposed. First, a [CNN](#)-based object detector [45] is used to localize objects in an image plane. In the second step, an [Autoencoder \(AE\)](#) predicts the 3D orientation of the object from the resulting scene crop.

The main contribution is a new training method for [AEs](#) that allows learning 3D object orientations from synthetic views of a 3D model, dispensing with the need to annotate orientations in real sensor data.

The [AE](#) is trained to revert augmentations applied to the input and thus becomes robust against irrelevant color changes, background clutter and occlusions. It learns to produce low-dimensional representations of synthetic object orientations which can be compared to the representations of real RGB test data in a [k-Nearest-Neighbor \(kNN\)](#) search.

Experiments on the pose annotated dataset T-LESS [23] prove the performance of the approach on different sensors. Finally, the training on synthetic data is shown to be almost on par with the training on real data.

Table of Contents

1	Introduction	1
2	Related Work	5
2.1	Local feature-based approaches	5
2.2	Global descriptor/template-based approaches	6
2.3	2D object detection	8
2.3.1	Implications for object pose estimation	9
2.4	Datasets	10
3	Estimation of 3D orientation	13
3.1	The representation of 3D orientations	13
3.1.1	Orientation regression	13
3.1.2	Object Symmetries	14
3.1.3	Classification	16
3.1.4	Pose Annotations	16
3.2	Autoencoders	17
3.2.1	Plain Autoencoder	17
3.2.2	Variational Autoencoder	19
3.3	Modeling and training the Variational Autoencoder	22
3.3.1	Model	22
3.3.2	Training Overview	24
3.3.3	Augmented Training Procedure	26
3.3.4	Training embedding generation and testing procedure	29
3.4	Multiple Objects	31
3.5	Inference Time	32
4	Experiments and Discussion	33
4.1	Evaluation Metrics	33
4.1.1	Axis-angle Rotation Error	33
4.1.2	Visible Surface Discrepancy	34
4.1.3	Mean Average Precision	34
4.2	Single Shot Multibox Detector	35
4.3	VAE Training Conditions	37

4.4	VAE Evaluation Results on T-LESS	37
4.4.1	Training on Canon images	37
4.4.2	Training on synthetic views	39
4.4.3	Latent Space Size	43
4.5	Analyzing the Latent Space	44
4.6	Variational Autoencoder vs. plain Autoencoder	45
4.7	Joint object detection and 3D orientation evaluation	46
4.8	Variants	48
4.8.1	Conditional Variational Autoencoder	48
4.8.2	Training on CAD models	50
5	Future work	51
6	Conclusion	53
	Bibliography	55
	Figures	61
	Tables	63
	Appendix	65

1 Introduction

Mobile industrial robots and service robots need to interpret dynamic and unknown environments. Many traditional perception approaches are tuned for fixed environments and fail in more general settings. These failures are critical and must be addressed since they limit the areas of application. Autonomous robotic behaviors require rich and reliable models of the robot's surroundings that are robust in all conceivable conditions. A large number of dangerous, repetitive or assisting tasks could potentially be solved if these limitations are overcome.

Many applications such as robotic manipulation and assembly, augmented reality and autonomous driving require the position and orientation of nearby objects. In Figure 1.1 DLR's robot Justin is attempting to grasp a cup. Therefore, it needs to accurately determine the pose of the cup as well as the pose of its end-effector hands. Without this information physical interactions and high-level scene interpretation are hardly possible.

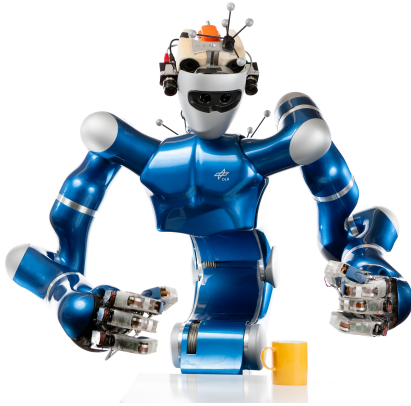


Fig. 1.1: DLR Justin

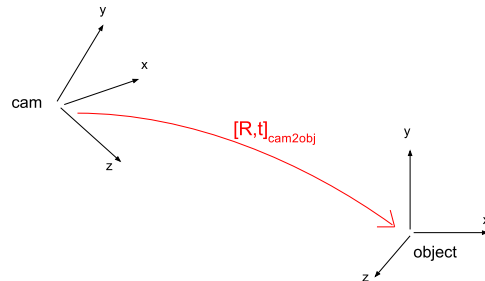


Fig. 1.2: Coordinate transformation

6D object pose estimation is the task of finding the 3D translation and 3D rotation from a camera to an object. Figure 1.2 sketches the corresponding homogeneous transformation. The goal is to recognize one or more known objects from a single scene and determine their positions and orientations efficiently. It constitutes an important subtask of robotic perception. Commonly, color (RGB), depth information or a combination of both is used for this process.

The field evolved from pick-and-place systems that used cameras to locate objects in static, fenced in environments. Driven by new applications, the demand and requirements have steadily grown. Today, it is one of the main research areas in computer vision.

Depending on the use-case, the requirements related to runtime and accuracy of pose estimation algorithms differ substantially. In case of augmented reality we seek for real-time capabilities while decreased accuracy is often sufficient to satisfy human cognition [39]. On the other hand, autonomous robotic assembly need a very precise perception of the manipulated objects [8]. Ideally, both goals can be met at the same time. However, visual perception is often the temporal bottle-neck of robotic tasks.

The main challenges in 6D pose estimation of rigid bodies can be summarized as:

Foreground occlusions: incomplete and disturbed representations

Background clutter: object - background distinction

Huge 6D search space: millions of possible 3D orientations + 3D translations

Object symmetries: ambiguous or almost ambiguous poses

Environment/Sensor changes: differences in hue, saturation, light, noise,..

Texture-less objects no recognizable texture for feature extraction

Multi-instance objects: multiple object instances in one scene

Limited data availability: 3D models and pose annotated viewpoint data

Many of these problems have been individually addressed in previous works. However, even the most complex approaches often rely on strong assumptions related to object characteristics, environment conditions or available data. Therefore, they can only be applied to a narrow range of scenarios. In consequence, object pose estimation inside operating robots still frequently depends on simple but non-general methods including AprilTags [44], table top segmentation or color blob detection.

A main issue in designing efficient and robust pose estimation algorithms is the size of the 6 dimensional geometric search space. Considering a moving camera around an object, there exist millions of possible views even when the dimensions are sampled at rather broad intervals. Naive template-based approaches are therefore computationally intractable. To decrease the number of combinations, the problem can be divided into individually estimating the image-plane translation, depth and 3D orientation of the object.

In recent years, a concept that addresses the lack of generality has revolutionized computer vision. **CNNs** learn to extract complex features in unstructured data such as images from the demonstration of examples. Through their high dimensionality and hierarchy, the expressiveness of **CNN** features is often only limited by the versatility of presented data. They are state-of-the-art in high-level vision tasks such as object classification [29], detection [24] and segmentation [15].

In 3D orientation estimation **CNN** architectures have not had similar breakthroughs, yet. One reason is that most supervised deep learning algorithms require great amounts of labeled data

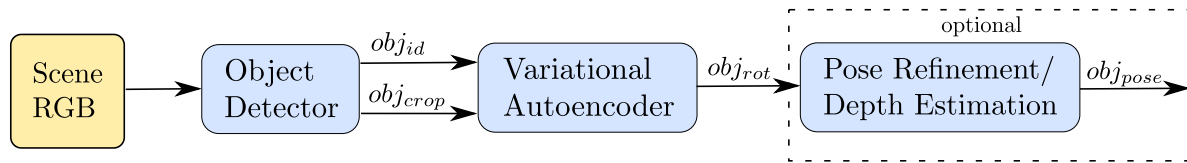


Fig. 1.3: Proposed object pose estimation pipeline

for training. However, generating pose annotated image data requires expert knowledge and a complex setup. Furthermore, the training data is usually produced in controlled environments which increases the risk of over-fitting.

In comparison, generating 2D bounding box annotations in images to train object detection algorithms is relatively easy. Object detection, i.e. localizing and classifying objects in an image plane, can be seen as a subtask of object pose estimation, reducing the 6D search space by two dimensions. Recently, a variety of CNNs has been introduced that efficiently solve this task even under the challenging conditions described above [48, 36, 37].

Therefore, in this work, the [Single Shot Multibox Detector \(SSD\)](#) [36], a CNN for object detection, is trained to locate and classify all considered objects in a scene image. The object detection is depicted as the first step of a pose estimation pipeline in Figure 1.3. The predicted bounding boxes are utilized to produce object crops from the scene image.

In a second step, these crops are served to a CNN-based AE architecture to obtain the 3D object orientation. Typically, AEs are neural networks that can be trained to learn low-dimensional encodings of the input data. This thesis presents a novel approach to steer AEs to efficiently produce codes that strongly correlate with the 3D orientations of an object in test image crops.

Why Autoencoders? In machine learning, problems are usually categorized into classification or regression. Treating the 3D orientation estimation as a classification problem would only be feasible for large sampling intervals as otherwise the number of classes and thus the number of parameters in the last fully connected layer would explode. On the other hand, the direct regression of $SO(3)$ representations like rotation matrices or quaternions is difficult due to representational ambiguities or constraints and due to indistinguishable orientations of symmetric objects or views. These issues will be described and discussed in detail in chapter 3.

Instead of regressing fixed orientation representations, this work proposes to learn a representation which only depends on the appearance of different object orientations. For this purpose, an Autoencoder CNN architecture is trained to reconstruct input images depicting an object at random 3D orientations. The input is first encoded to a lower dimension, then decoded to the original input size. After training, the low-dimensional bottle-neck of the AE can be used as a descriptor for an input image. Since the object appearance is tightly coupled with

the object orientation, similar descriptors usually originate from similar orientations. At test time, the encoder can produce a descriptor from a scene image crop containing the object. In a Nearest Neighbor search, the next training descriptor whose corresponding orientation is known determines the object orientation in the test image.

Our proposed solution to cope with the limited data availability is to learn the object orientation from synthetic camera views of textured 3D models.

Today, hand-held RGB-D sensors can be used to produce accurate, textured 3D model reconstructions. By approximately orbiting the device around an object, algorithms like KinectFusion [42] can fuse the recorded sequence to a consistent 3D model. This enables the production of large datasets of surface 3D models [4]. The 3D reconstructions can be used as an infinite data source of pose annotated, virtual views.

The remaining challenge is the generalization from synthetic training data to real RGB test data. The test data distribution is clearly different to the training data distribution. Therefore, there is a great risk of over-fitting to features in the synthetic training data that do not exist in the same form in the test sensor data. Furthermore, the test scene crops might include unseen background and foreground occlusions which disturb the representation.

The AE architecture can explicitly be trained to become invariant against these differences. This is achieved by applying strong augmentations at the input side of the AE but letting the decoder reconstruct clean object views. This method is inspired by the denoising Autoencoder [58] which takes noisy images as input and reconstructs a clean version of the input image. Here, the augmentations ensure that the descriptors become invariant against irrelevant color differences, lighting conditions, background clutter and foreground occlusions.

The augmentations have a key role to make this method work. The result is a robust, real-time 2D detection and 3D orientation estimation pipeline.

2 Related Work

There exists a vast amount of literature about object pose estimation. In this chapter I will briefly categorize different approaches and describe their advantages and disadvantages. Of course, the perfect fit depends on a number of prerequisites. Can we obtain appropriate data for training? How does the environment look like and what sensors depict it? What are the time constraints? The following approaches were designed with different scenarios in mind.

Although the boundaries sometimes overlap, pose estimation algorithms for rigid bodies can be divided into *template-based* and *local feature-based* methods. Some techniques require a learning phase and others only need to extract features in advance.

2.1 Local feature-based approaches

Local feature points represent image patterns such as corners or edges. More complex features like the [Scaled Invariant Feature Transform \(SIFT\)](#) and [Speeded Up Robust Features \(SURF\)](#) also come with a descriptor of the surrounding patch which can be compared to other descriptors in a database. Most feature-based pose estimation methods identify 2D-3D feature correspondences between a textured 3D model and a scene image [\[38, 6\]](#). Using a [Perspective-n-Point \(PnP\)](#) algorithm [\[35\]](#) at least four spatial correspondences yield a definite pose estimate, assuming the intrinsic camera parameters are known. In practice, several hundred of these features are found and a computationally intensive algorithm like [RANSAC](#) has to be used to eliminate outliers.

While local feature-based approaches work quite well on textured objects where unique feature positions can be found, the performance drops significantly on untextured objects where the feature positions are ambiguous. Furthermore, the features are mostly handcrafted and thus are sensitive to perturbations such as contrast changes and clutter. For that reasons the research community recently shifted to learning robust features. Most of these approaches utilize [Random Forests \(RFs\)](#) or [CNNs](#).

Crivellaro et al. [\[5\]](#) predefine 3D object coordinates and manually label their respective projections in a set of training images. Using a [CNN](#) for 2D point detection, the authors regress the positions of the projected 3D points in the image plane. At test time, the key-point positions in a scene image can be determined which enables the use of a [PnP](#) algorithm for pose estimation. Their method is robust to partial occlusions and a variety of image perturbations.

Unfortunately, the generation of the training set is costly and the approach has not been evaluated against public datasets, yet.

Recently, Rad and Lepetit [47] presented a similar method by matching 2D-3D correspondences of 3D bounding box corners. These coordinates are only available if training images with registered 3D models are available. The authors build a classifier to manually distinguish object symmetries and in total their approach consists of 5 CNNs which makes it computationally expensive. On the LineMOD dataset [20] they claim state-of-the-art performance in the RGB-only domain.

In the RGB-D domain most local-feature approaches rely on patch descriptors or densely sampled features. The reason is that depth point distributions are often rather coarse, especially if the object is further away.

Brachmann et al. [2] introduce a dense feature-based approach where a RF separately predicts for each RGB-D pixel the object class and coordinate on a 3D object. For training, segmented objects and arbitrary background RGB-D data are required. The advantage of this method is that it works accurately even under moderate occlusions. After an average of 160ms around 200 pose hypotheses are obtained. The following pose optimization step depends on a RANSAC algorithm and takes about 400ms on a powerful CPU. While the optimization is crucial for their accurate pose estimation, it slows down the whole pipeline significantly. The work was extended to articulated [41] and deformable [30] objects as shown in subsequent papers.

Kehl et al. [25] also uses an Autoencoder to learn descriptors from random local patches of real RGB-D data. Then they use the Autoencoder to create a descriptor database for each synthetic view of a textured 3D model together with the corresponding 6D pose. At test time, descriptors are generated again from random patches in an RGB-D scene which vote for the 6D object pose. Depending on the local consistency of votes the final 6D object pose is determined. The approach requires to evaluate a lot of patches in a scene and therefore takes about 670ms per prediction.

Local feature points or patches circumvent the extensive search in the 6D pose space and are robust to occlusions. However, spatial relations between the features are hardly exploited. They are at risk to miss small artifacts that distinguish object poses. Dense feature-based approaches address this issue but usually require a time-consuming RANSAC step.

2.2 Global descriptor/template-based approaches

In contrast, global template-based approaches determine a description of the whole object from different viewpoints. Afterwards, these holistic templates are matched against a scene where a similarity measure produces the most likely pose. The descriptor can either depict a template image of the object that can be spatially matched to the scene or an abstract low-dimensional vector which can be matched in a kNN search.

Some approaches rely on recognizing annotated real recordings of objects in test scenes. Others only use synthetic data generated from 3D models. The latter is favorable since pose annotated data is rarely available outside dataset science.

The LineMOD algorithm by Hinterstoisser et al. [18] is a template-based method that was gradually improved to one of the most efficient learning-free methods. They generate multi-modal templates from a reconstructed 3D model that include color gradients and depth surface normals. By carefully tuning these handcrafted templates they achieve impressive performance on their own LineMOD dataset [20]. The greatest advantage is that due to their efficient implementation, the runtime is only about 120ms on a CPU, not far from real-time. However, Brachmann [2] shows that the performance drops significantly under different lighting conditions. The RGB-only version is called Line2D [19] which works considerably less well.

Ulrich et al. presented a method [57] where 3D edges from CAD models are extracted at different viewpoints and matched against a monocular scene image. To be robust against lighting changes, the similarity measure is only based on the dot-product between normalized edge gradient directions. Their contribution is an efficient hierarchical search at different resolutions and viewpoint intervals. That way, the exhaustive search over the viewpoint space can be accelerated. Still, additional assumptions about the possible search space are necessary to make this method applicable to most practical use-cases. Otherwise, the computational effort is intractable.¹

A recent successful approach from Wohlhart and Lepetit [62] suggests to learn global descriptors to discriminate objects and their 3D orientations using a CNN architecture. The network is trained on real and synthetic images of the LineMOD dataset using ground truth pose information. They propose a new loss function which consists of a triplet-wise loss term, a pair-wise loss term and a regularization term. The three training samples for the triplet loss are chosen as either / or

- 2 training samples with similar pose and 1 sample with different pose from same object
- 2 training poses from the same object and 1 sample from different object

The triplet loss maximizes/minimizes the L2 distance between the descriptors of dissimilar/similar samples. The pair-wise loss is computed on similar samples where one is clean and the other one is augmented by artificial illumination changes and noise. The L2 distance between both is minimized to make the descriptor robust against these disturbances. After training, descriptors are produced from clean, textured 3D models that are recorded from virtual cameras placed on a viewsphere (varying azimuth and elevation). Then a test scene crop is fed into the CNN producing a test descriptor for whom the k nearest virtual views can be found. The pose and class of the nearest neighbor can be used to evaluate the algorithm.

¹During my thesis I implemented a GPU-based version of this algorithm as a learning-free baseline

It remains open how the 2D object bounding box detection is performed. The authors assume it to be given just as the distance to the object (which is a reasonable assumption when we deal with RGB-D data). Data greediness is arguably the biggest drawback due to the reliance on real annotated sensor data. The performance is fast and robust, outperforming the LineMOD algorithm on the LineMOD dataset where objects lie upright on a table. It also scales well with the number of considered objects.

The method presented in this thesis has conceptual similarities to the approach of Wohlhart and Lepetit as they also learn low-dimensional descriptors of object views using a CNN. However, their approach is supervised and symmetries in objects have to be explicitly determined in advance.

2.3 2D object detection

2D object detection can be seen as a sub-problem of 6D object pose estimation. The goal is to regress the 2D bounding boxes of all objects in an RGB image while concurrently determining their classes. The 6D pose estimation problem can thus be reduced to search a particular object at a particular spot in the scene image. It remains to estimate the depth and the 3D rotation from the camera to the object.

Object detection is a well-studied problem in literature. Early approaches focused on certain domains like face and pedestrian detection so that they could hand-design appropriate features. Viola et al. [59] achieved near real-time performance in object detection using a boosted cascade of simple features. However, in recent years many computer vision challenges like ImageNet [50] have proven that CNN-based approaches yield the most accurate and general results in this domain. While the competition goal is to obtain the highest accuracy and consequently all solutions are composed of big model ensembles, in a real world scenario runtime and memory consumption are equally important.

Among the state-of-the-art practical detectors are namely FasterRCNN [48], the Single Shot Multibox Detector (SSD) [36] and R-FCN [37]. What all three approaches have in common is a base-network that acts as a feature extractor. The feature extractor consists of a standard convolutional neural network architecture where all fully-connected layers are removed. Consequently, when feeding a high resolution image into the network, the resulting feature maps after each convolutional layer correspond spatially to the input image. That way the features for the whole image can be efficiently computed together and the predictions are based on features appearing in a specific input region. For the feature extractor usually very deep architectures as VGG-16 [53], ResNet101 [14] or Inception v3 [56] are used.

The heads ("meta-architectures") of the networks differ in how they interpret the feature maps. Without claiming completeness, the major distinctions are briefly described in the following.

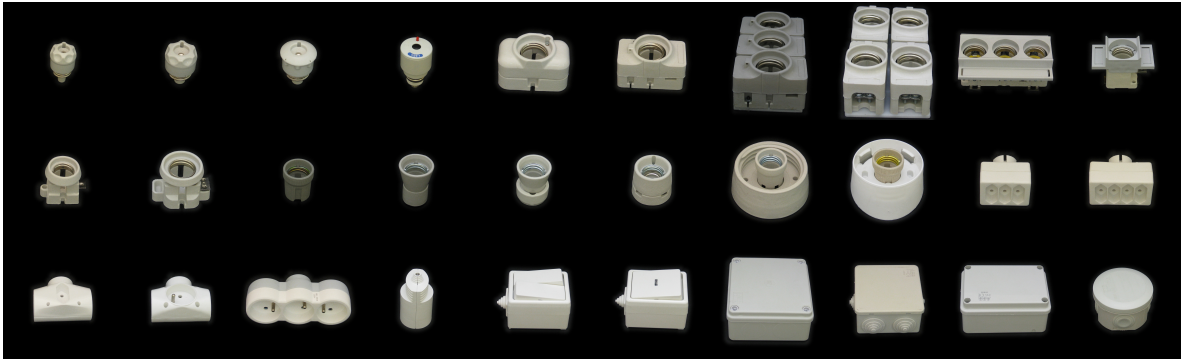


Fig. 2.2: All 30 T-LESS objects in ascending order

Empirically, R-FCN is a trade-off between accuracy and runtime but FasterRCNN can be just as quick at similar performance when reducing the number of box proposals. SSD is the fastest approach to predict all boxes in low resolution images (300x300). It achieves comparable accuracy, especially on bigger objects.

The information what objects exist in which part of the scene projection is already quite valuable for robotic applications and strongly facilitates the subsequent pose or orientation estimation step. In the context of 3D orientation estimation of rigid objects I decided to finetune SSD with VGG16 to use it as the first module of the pipeline. The decision for SSD is motivated by the following:

- rigid objects are less of a challenge for modern object detectors
- runtime of $\sim 17\text{ms}$ vs. $\sim 142\text{ms}$ FasterRCNN, both with VGG16, while achieving a competitive mAP [36]
- SSD's weakness with very small/distant objects is neglectable in robotic manipulation
- SSD is conceptually simpler and allows a straight forward transfer learning scheme

In the evaluation I will describe the implementation details and results of this method using different kinds of data, including synthetic RGB renderings.

2.4 Datasets

To evaluate pose estimation algorithms, there exist a small number of datasets offering pose annotated data from different RGB and RGB-D sensors. Hinterstoisser et al. presented the LineMOD dataset [20] which has been a popular baseline in the past years. It provides view-sphere RGB-D data of scenes and single objects as well as 3D meshes for all 15 objects. The dataset contains challenging clutter and occlusions, but the objects do not exhibit symmetries meaning that poses are definite. However, in industrial or household environments symmetri-

cal or almost-symmetrical objects are common and many algorithms suffer from these pose ambiguities.

To meet this issue, the recently released T-Less dataset [23] presents 30 industrial-relevant objects of whom several contain different kinds of symmetries. They were recorded using 3 modern sensors: a Microsoft Kinect v2, a Canon IXUS 950 IS camera and a Primesense Carmine RGB-D sensor. Therefore, the presented approaches are evaluated on T-Less.

The first SIXD Challenge [21] for object pose estimation in October 2017 will make the results from literature more comparable. The organizers converted six publicly available pose annotated datasets to a standard format for facilitated access and evaluation. I will measure my results with the data and error metrics proposed in this challenge.

3 Estimation of 3D orientation

Starting from 2D object detections in an RGB image, this chapter describes a new approach to determine 3D object orientations. The result will be a real-time object detection and 3D orientation estimation pipeline trained on synthetic views of 3D models.

To further motivate the presented method, the issues in supervising 3D orientation estimation algorithms are first explained.

3.1 The representation of 3D orientations

Orientation estimation is a hard problem since the group of 3D rotations $[SO(3)]$ depicts a large, non-Euclidean space. In this section, the issues related to different $SO(3)$ representations are explained. Standard classification and regression techniques that are popular in other domains suffer from these issues.

Simple approaches build up experience and often lead to unexpected findings. In the case of learning a 3D orientation from RGB or RGB-D images, one naive learning approach would be to feed pose annotated object images into a high capacity neural network and regress, for example, the three Euler angles (α, β, γ) . Alternatively, we could create a class for each discrete object view and train a one-hot orientation classifier. However, in literature there exist only some classification and very few direct, template-based regression methods [52].

3.1.1 Orientation regression

The first issue of directly regressing Euler angle data stems from the periodicity, i.e. the wrap up after a full rotation cycle. For a 1D rotation $\alpha + 360^\circ = \alpha \mid \alpha \in \mathbb{R}$. Even if we restrict to $\alpha \in [-180^\circ, +180^\circ)$ the regression task still suffers from a discontinuity at $\pm 180^\circ$. Lets consider a training sample target that is near the discontinuity $\alpha = 179^\circ$. It might be that the estimate only slightly misses the regression target in its physical meaning, e.g. $\hat{\alpha} = -178^\circ$, but it is very strongly punished. In practice, this kind of regression formulation usually learns an expected angle of $E[\alpha] = 0$. If we optimize the regression error based on the true angular difference $\min(|\alpha - \hat{\alpha}|, |\alpha - \hat{\alpha} - 360^\circ|)$ we can run into the problem of steering the regressed value to the limit of the restricted domain $\hat{\alpha} \rightarrow \pm 180^\circ$.

A proposed solution is to learn the Wrapped Normal distribution [11] which acknowledges the circular nature of the problem. The idea is to fit a Gaussian model that explains the angular data distribution using an [Expectation Maximization \(EM\)](#)-algorithm. However, this approach only works for independent random variables, i.e. only for 1D rotations. The 3D Euler angles are interdependent so that they can not be learned separately.

Another problem with Euler angles is the so-called Gimbal Lock. There are orientations where two of the three Euler rotation axes align. In these settings there is no unique solution, meaning that infinite angle combinations depict the same orientation. Near a Gimbal Lock a small change in orientation can correspond to a large change in Euler angle space. Overall, it can be seen that Euler angles are not suitable for a full 3D orientation regression.

A rotation matrix has a unique representation for every orientation. Since rotation matrices are orthogonal $SO(3) = \{R \in \mathbb{R}^{3 \times 3} \mid R^T R = R R^T = I\}$ and $\det(R) = 1$, they have [six degrees of freedom \(6DOF\)](#). If two axes are known the third one is uniquely defined. The main difficulty when regressing rotation matrices is to ensure the orthogonality constraint between the axes. Direct regression of rotation matrices is therefore uncommon.

Unit quaternions ($q \in \mathbb{R}^4, \|q\|_2 = 1$) can also represent 3D orientations. The elements vary continuously over the \mathbb{R}^4 unit sphere and thus, Gimbal Locks are not an issue. However, quaternions still exhibit an ambiguity in their representation of a single orientation. More precisely, q and $-q$ depict the same rotation. It is therefore necessary to neglect the possible confusion of the learning algorithm by the other representation. For quaternions only the normality constraint must be satisfied which is simpler to ensure. Recently, Doumanoglou [7] managed to train a siamese CNN architecture similar to [62] with additionally introducing a loss that demands the L2 distance between the features to correspond to the L2 difference between the quaternions. This can be interpreted as an indirect orientation regression. Kendall et al. [26] introduced PoseNet where the quaternion orientation of the camera with respect to the world is regressed. This is a simpler problem because features can be found in the whole image. Overall, a successful example of direct quaternion regression for 3D object orientation estimation could not be found in literature. Additionally, simple experiments of a CNN-based regression on quaternions conducted by ourselves failed.

3.1.2 Object Symmetries

Apart from the discussed representational ambiguities, template-based regression is hardly possible if dealing with symmetric objects.

Symmetries are a challenge for many object pose estimation algorithms. In particular, a symmetric object has from two up to infinite ambiguous orientations. All of these orientations can be correct but without further treatment there is only a single correct regression target. Only if the symmetries are known and explicitly defined in advance, a regression algorithm can

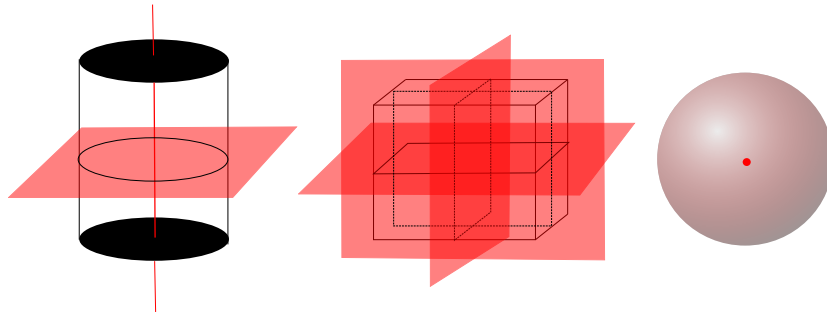
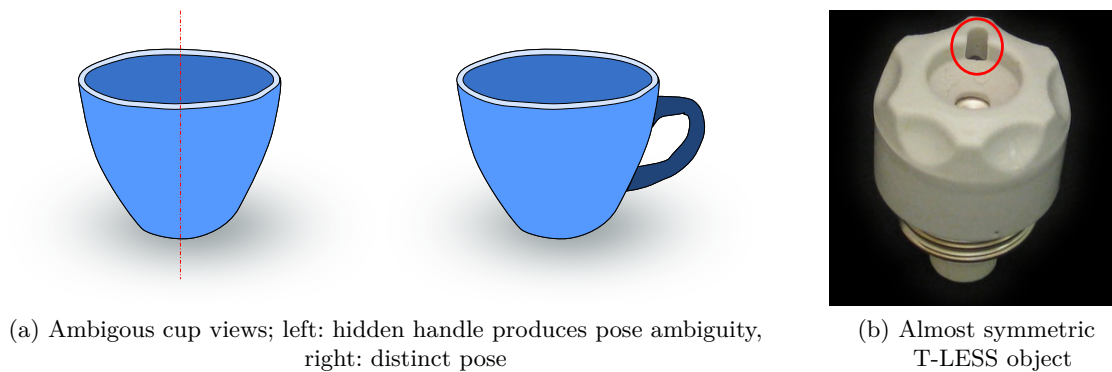


Fig. 3.1: Overview of different object symmetries: axis+plane, 3-planes, point

account for this issue. Symmetries can appear in multiple combinations around axes, planes and points. Some examples are depicted in Figure 3.1.

Even if all object symmetries are automatically detected, there is still the issue of indistinguishable views of objects which are not necessarily symmetric themselves. A popular example is a cup where at certain poses the handle is not viewable and thus the recordings appear the same. Figure 3.2a shows such a view-dependent axis-symmetry. Many objects in household, industry or automotive environments exhibit this kind of symmetry. Likewise, partial occlusions can cause pose ambiguity.

Finally, an object might contain only minor hints regarding its definite orientation. In some cases as depicted in Figure 3.2b, only the texture or tiny artifacts distinguish the views. In this situation, one has to either rely on the regression algorithm or mark a symmetry in advance. The decision also depends on the considered task. For successful grasping of objects using a robot, the actual orientation might be irrelevant. However, if autonomous robotic assembly is the goal, object orientation is often crucial.



(a) Ambiguous cup views; left: hidden handle produces pose ambiguity, right: distinct pose

(b) Almost symmetric T-LESS object

Fig. 3.2: Symmetric and almost symmetric views

3.1.3 Classification

Another valid pose estimation approach is to divide the orientation space $SO(3)$ into discrete views by using an equidistant quaternion interpolation or simply discretized Euler angles. Note that for classification the above mentioned ambiguities are not as severe. However the number of discrete views rises cubically with the sample frequency. A classification algorithm that disregards in-plane rotations and only estimates azimuth and elevation can work well, as will be shown in chapter 4. However, considering all possible rotations at coarse 5° sampling intervals results in over 50000 classes. CNNs and Multilayer Perceptrons (MLPs) usually perform classification after a final fully-connected layer that maps the input to an output of a dimension corresponding to the number of classes. Consequently, the number of trainable parameters explodes and despite heavy data augmentation there are presumably not enough training samples per class. This makes naive classification for the full 3D rotation space unsuitable.

3.1.4 Pose Annotations

One of the most crucial obstacles, that prevents the practical usage of machine learning algorithms in general, is the lack of labeled data. While class and even 2D bounding box labels are moderately cheap to generate by humans, accurate pose annotations of real sensor data requires great effort by computer vision experts. Usually, this involves a complex acquisition setup where a turn-table surrounded by markers is recorded at different tilt and azimuth angles (e.g. T-LESS [23]). Alternatively, a robot arm can be used to orbit around an object on a half-sphere executing recordings at predefined positions (e.g. THR dataset []). Both approaches are difficult and cumbersome. Hence, many practitioners fall back to less robust and accurate but learning-free methods like CAD-based edge matching [57].

A goal of this thesis is to show that data-assisted pose estimation is possible without doing manual pose-labeling. As mentioned in the introduction, the latest innovations in 3D reconstruction allow to build accurate, textured 3D model from sequential RGB-D data of hand-held devices [43]. In chapter 2, state-of-the-art approaches that extract local features [2] or whole templates [18] from these reconstructions to match them against real scenes for pose estimation have already been presented. However, local features are independent from each other and do not exploit the entire context information. On the other hand, template-based approaches have been shown to be less robust against occlusions. To overcome the limitations of templates, we will incorporate the knowledge of possible occlusions into the training data.

3.2 Autoencoders

Autoencoders are a self-supervised learning technique. This work will describe how they can be adjusted to disentangle the 3 dimensional rotational space from different kinds of real and synthetic data. The goal is to quickly obtain object pose hypotheses from a scene that are robust to occlusion, lighting changes, sensor types and other disturbances. Basically, the idea is to train a low-dimensional descriptor for every (discretized) 3D orientation. At test time, the same network generates a test descriptor. The nearest training descriptors have orientations assigned that are, in the best case, very near to the test object’s ground truth orientation.

We have seen that ambiguities from object symmetries represent challenges for many known pose estimation and tracking algorithms. Consider for example the previously mentioned work from Wohlhart et al. [62] which attracted quite some attention. The conceptual goal of our approach is similar in the sense that we build an abstract descriptor space from templates. However, their triplet loss assumes that dissimilar poses can be separated in the descriptor space. For symmetric objects or views, where dissimilar poses basically look the same, this approach is likely to fail. In contrast, the AE creates the descriptor space solely based on the appearance of the objects. If there is no difference in appearance it will not attempt to push the descriptors away from each other. Thus, the AE automatically abstracts from symmetric ambiguities and is unaffected by representational ambiguities.

3.2.1 Plain Autoencoder

In this section, the theory and applications behind the plain Autoencoder are introduced.

The original Autoencoder, introduced by Hinton et al. [49], is a dimensionality reduction technique for high dimensional data such as images, audio or depth. It consists of an encoder and a decoder, both arbitrary learnable functions. In recent years, they commonly have the form of fully-connected or convolutional layers because of their high capacity towards complex data structures. In between encoder and decoder there is a low-dimensional bottleneck which is often called the latent space $Z \in \mathcal{R}^n$ with typically $1 \leq n \leq 200$. Autoencoders are a self-supervised learning technique, i.e. the desired output y is defined as the input:

$$y_{(i)} = x_{(i)} \tag{3.1}$$

The objective is to reconstruct the inputs $x_{(i)} \in \mathcal{R}^{\mathcal{D}}$ with dimension \mathcal{D} from a latent code $z_{(i)} = \text{Encoder}(x_{(i)})$:

$$\hat{x}_{(i)} = \text{Decoder}(\text{Encoder}(x_{(i)})) \tag{3.2}$$

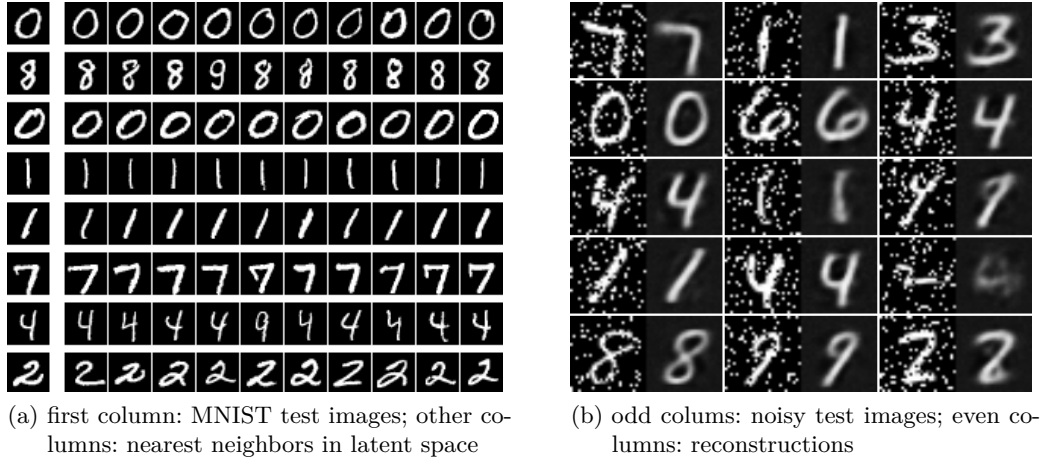


Fig. 3.3: Autoencoders for clustering and denoising

Therefore, the back-propagated loss simply consists of a sum over the pixel-wise L2 distance

$$L_{(i)} = \sum_{\mathcal{D}} \|x_{(i)} - \hat{x}_{(i)}\| \quad (3.3)$$

The latent codes from all training samples are referred to as the training set embedding. After training, it is likely that latent codes from samples with similar appearance are close to each other. Therefore, Autoencoders can be used for unsupervised clustering. For latent codes of test samples, the nearest training codes let us predict their characteristics or class affiliation. If semantic labels are available, the classification results are obviously inferior to supervised learning. Nonetheless, regarding the MNIST dataset [34] results are still satisfying as can be seen in Figure 3.3a. Note that the Autoencoder does not explicitly learn the class affiliation of a number but rather everything that composes the appearance like orientation, stroke diameter and even style. Especially its ability to separate orientations as in row 4 and 5 will prove to be useful in the pose estimation task.

Besides compression and clustering, one application of Autoencoders is denoising. To train a denoising Autoencoder [58], the input is perturbed with random noise while maintaining the desired output clean. After training, the model is able to clean noisy test images. Figure 3.3b depicts a denoising AE trained on noisy MNIST digits.

In fact, the denoising Autoencoder violates the original notion of using the exact same input and output. However, this particular method shows that you can weakly supervise Autoencoders using 'augmented input / clean output' pairs in order to become invariant against artificial augmentations. In this context, 'invariant' means that the encoder should produce the same latent codes for the same image with different pixel perturbations. This is crucial for the decoder performance as otherwise any perturbation that was not seen during training would possibly result in unclear latent codes which prevent a clear reconstruction and attri-

bution. This behavior is later used to weakly supervise the VAE training to obtain robustness against a large variety of perturbations that can occur in object pose estimation.

3.2.2 Variational Autoencoder

After this quick introduction of the Autoencoder, the Variational Autoencoder (VAE) is now introduced. The motivation is to learn a meaningful latent representation that is capable of depicting 3D orientations.

The [Variational Autoencoder \(VAE\)](#) [28] belongs to the class of Bayesian Neural Networks. Instead of learning arbitrary numerical values as latent codes, the VAE constrains the encoded representations to constitute a statistical latent variable model. The objective is to determine the output probability distribution $P(X; \theta)$ that best explains every sample of the input data X using an Autoencoder (neural network) parameterized by θ . The decoder can be interpreted as a joint distribution $P(X|z; \theta)$ that describes the probability of X given a latent variable $z \sim P(z)$ and the network weights θ . The encoder is equivalently $P(z|X; \theta)$, i.e. the probability distribution of a latent variable for some input data. By integrating over all possible z and marginalizing z out of $P(X, z)$, we find the relation:

$$P(X) = \int P(X, z; \theta) \, dz \stackrel{\text{marg.}}{=} \int P(X|z; \theta) P(z) \, dz \quad (3.4)$$

How is the model trained to maximize the likelihood of reconstructing the input data? $P(X|z; \theta)$ is unknown but we could use samples from z to approximate it as

$$P(X) \approx \frac{1}{n} \sum_{i=0}^n P(X|z_{(i)}; \theta) \quad (3.5)$$

However, this is computationally very expensive and wasteful since most $z_{(i)}$ will generate $P(X|z_{(i)}; \theta) \approx 0$. The idea is to learn a simple probability function like a (multivariate) Gaussian $Q(z|X)$ whose samples $z_{(i)}$ are likely to produce $P(X|z_{(i)}; \theta) \gg 0$. Then only few examples were necessary to relate $P(X|z; \theta)$ to $P(X)$ which enables the optimization. θ is omitted for brevity in the following.

To approximate $Q(z|X)$, the so-called [Kullback-Leibler \(KL\)](#) divergence is used. The KL-divergence describes the difference between two probability distributions. In this case, we force the encoder $P(z|X)$ to produce a multivariate Gaussian $Q(z|X) \sim \mathcal{N}(\mu, \Sigma)$ with mean μ and diagonal covariance matrix Σ . In practice, this means that the encoder outputs a mean and a variance vector for every input. Formally, the [KL](#)-divergence is minimized:

$$\mathcal{D}_{\mathcal{KL}}[Q(z|X) \| P(z|X)] = E_{z \sim Q}[\log(Q(z|X)) - \log(P(z|X))] \quad (3.6)$$

Minimizing the log probabilities is equivalent to minimizing the original probabilities due to the monotonicity of the logarithm operator. The Bayes Rule states:

$$P(z|X) = \frac{P(X|z)P(z)}{P(X)} \quad (3.7)$$

$$\Leftrightarrow \log(P(z|X)) = \log(P(X|z)) + \log(P(z)) - \log(P(X)) \quad (3.8)$$

Inserting eq. 3.8 into eq. 3.6 yields

$$\mathcal{D}_{\mathcal{KL}}[Q(z|X)||P(z|X)] = E_{z \sim Q}[\log(Q(z|X)) - \log(P(X|z)) - \log(P(z))] + \log(P(X)) \quad (3.9)$$

$\log(P(X))$ does not depend on z and was thus taken out from the expectation. Substituting $E_{z \sim Q}[\log(Q(z|X)) - \log(P(z))]$ with the definition of the KL-Divergence and rearranging the terms finally yields

$$\log(P(X)) - \mathcal{D}_{\mathcal{KL}}[Q(z|X)||P(z|X)] = E_{z \sim Q}[\log(P(X|z))] - \mathcal{D}_{\mathcal{KL}}[Q(z|X)||P(z)] \quad (3.10)$$

Recall that we want to maximize $P(X)$ or equivalently $\log(P(X))$ with respect to θ . The KL-divergence is always greater than zero. Thus, maximizing the right side of eq. 3.10 corresponds to maximizing a lower bound for $\log(P(X))$. But how does this term translate into a loss function? If we constrain $P(z)$ to be unit Gaussian, we can compute a closed form solution of $\mathcal{D}_{\mathcal{KL}}[Q(z|X)||P(z)]$. Without further derivation the KL-divergence loss states

$$L_{KL} = \mathcal{D}_{\mathcal{KL}}[\mathcal{N}(\mu(X), \Sigma(X))||\mathcal{N}(0, I)] = \frac{1}{2} \sum_n (\Sigma(X) + \mu^2(X) - 1 - \log(\Sigma(X))) \quad (3.11)$$

where n is the dimension of the latent space. This term could be interpreted as a regularization that makes the latent space behave like a unit Gaussian distribution. Other choices are possible but this one increases the pressure on the encoder $Q(z|X)$ to relate the latent codes in a dense latent space. Furthermore, it introduces a measure of uncertainty that is depicted by the variance of each latent variable.

Maximizing the first term $E_{z \sim Q}[\log(P(X|z))]$ has the opposing goal to create samples from $Q(z|X)$ that produce distinguishable latent codes which the decoder $P(X|z)$ can map back to the input. In consequence, to evaluate the first term we can simply use a pixel-wise L2 loss or cross-entropy loss on the reconstructions $P(X|z)$. $P(X|z)$ can be interpreted as a Bernoulli distribution that describes the probability of each pixel being one vs. being zero. Hence the reconstruction loss per sample is the cross entropy loss summed over all elements (pixels):

$$L_{rec} = - \sum_{\mathcal{D}} x \log(\hat{x}) + (1 - x) \log(1 - \hat{x}) \quad (3.12)$$

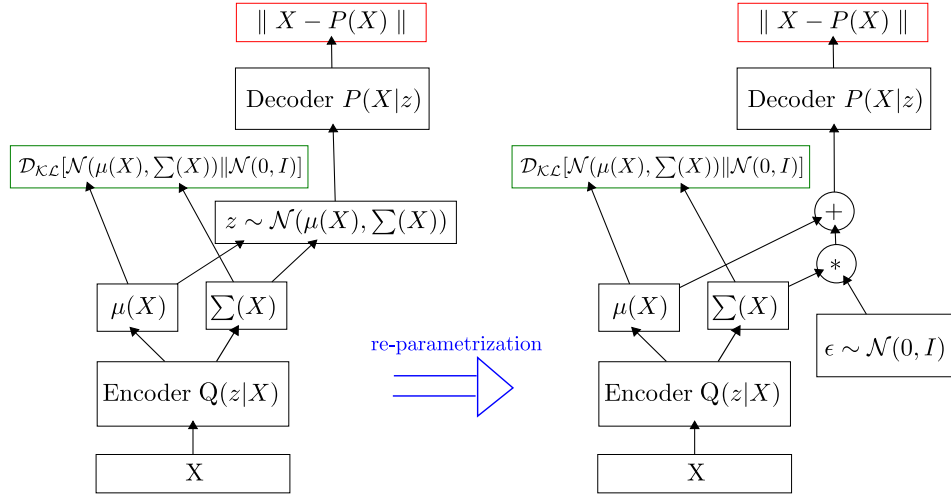


Fig. 3.4: Variational Autoencoder architecture and re-parametrization trick

The total encoder and decoder losses over a mini-batch with size K are then described by:

$$L_{enc} = \frac{1}{K} \sum_K L_{rec} + L_{KL} \quad \text{and} \quad L_{dec} = \frac{1}{K} \sum_K L_{rec} \quad (3.13)$$

A remaining problem is that the gradients can not pass through a non-deterministic sampling from a probability distribution during back-propagation. Fortunately, using the so-called 're-parametrization trick' we can model the sampling step as an added noise channel $\epsilon \sim \mathcal{N}(0, I)$ so that the gradients can pass through the whole network. See Figure 3.4.

Recently, Higgins et al. presented the β -VAE [16] which simply introduces a factor in front of the KL-Divergence term as a hyperparameter. Intuitively, it makes sense to control the ratio between the reconstruction loss and the KL-Divergence loss: The magnitude of the reconstruction loss is dependent on the input size \mathcal{D} and the KL-loss is dependent on the dimension of the latent space n . In extreme cases, one of the loss terms is dominant and the other one is ignored. The vanilla VAE does not deal with this issue.

3.3 Modeling and training the Variational Autoencoder

After the Autoencoder and its variants were theoretically derived, this section will focus on the network architecture and on novel ways to train and test the framework using different data sources.

An overall goal is to reduce the amount of memory and computation needed to execute a template matching in the full 3D orientation space. As previously mentioned, methods like hierarchical orientation estimation [57] at different resolutions, neglecting similar looking templates or applying domain knowledge can mitigate the issue of an exploding search space. However, in order to make the exhaustive search computationally practical, performing a strong dimensionality reduction for each sample view is essential. The objective of using the VAE is to create low-dimensional descriptors for each view in a densely sampled 3D rotational space.

3.3.1 Model

The implicit objective for the encoder is to find features in the input image with whom it can construct orientation dependent latent variables. The state-of-the-art feature extractors for image data are CNNs.

Therefore, the encoder depicted in Figure 3.5 consists of several convolutional layers with (4×4) filters and a stride (filter step-size) of two. The stride is responsible for downsizing the input by half after each convolutional layer so that features can be extracted at different scales and the input is subsequently reduced in spatial dimension. The resulting activations are sent through ReLU functions. Max-pooling is not used here, because accurate pose estimation relies on the exact positional information of the features and max-pooling lets the highest activation pass without regard to its position [55].

Contrary to the spatial dimension of the feature maps, the number of convolutional filters and thus the number of output feature maps are increasing with the depth of the encoder network, following common architectures as VGG16 [53]. Intuitively, the first convolutional layers need less filter variety since they only learn low-level features as color blobs, edges and corners.

After the convolutions, the last block of feature maps is flattened and fully connected to match the dimension of the mean and variance vectors. In case of the mean, no activation function is used so that it can take negative values. The variance activation function is a softplus $f(x) = \log(1 + e^x)$ which smoothly maps the domain of definition x to a positive codomain $f(x) > 0$. During training latent codes from the multivariate Gaussian $z \sim \mu + \Sigma * \epsilon$ are sampled as described in Fig. 3.4. At test time, $z = \mu$.

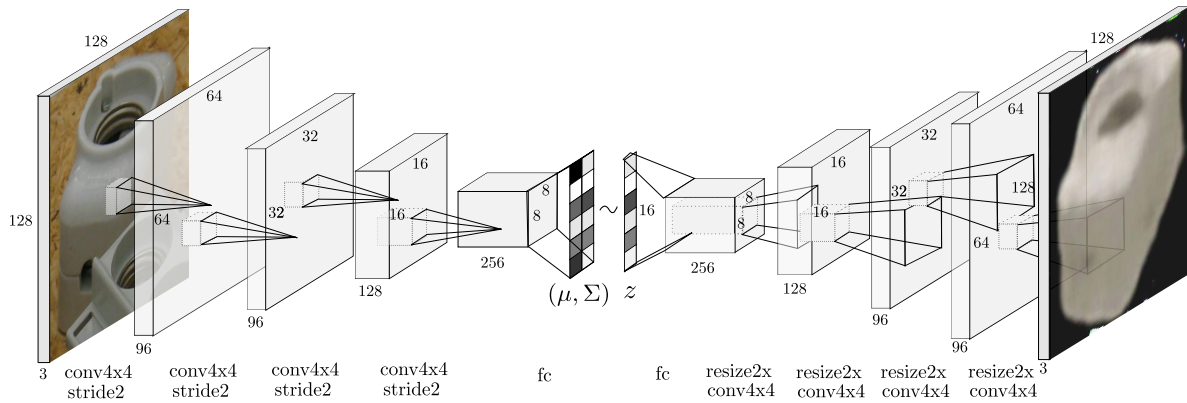


Fig. 3.5: Variational Autoencoder CNN architecture with input/output at test time; (simplified sampling, see Fig. 3.4!)

Since the bottleneck of the VAE, i.e. the latent space, is very low-dimensional, e.g. $\dim(z) = 16$, it is not possible to embed the whole appearance of the object view into the code. Without a larger memory, the reconstruction of complex image data is impossible. Therefore, the decoder has to memorize the appearance of the object, its features, their spatial relations and orientations. These information can very well be encoded into the weights of a CNN. After training, the latent code itself will activate the right combination of CNN filters to reconstruct a certain object view of the input image.

In (variational) Autoencoders the decoder is usually, but not necessarily, a mirrored version of the encoder. The notion behind this choice is that the same architecture should in principle be able to learn to decode the same type of features which it previously encoded. Therefore, the latent code z is transformed and reshaped back to a stack of feature maps using another fully connected layer. In order to transpose the convolutions we first perform a nearest neighbor upsampling of factor two on each feature map. Next, a convolutional layer of identical filter size (4×4) but stride 1 is applied. The last feature maps have the same extent as the input image, here $128 \times 128 \times 3$, and go through a sigmoid activation $f(x) = \frac{1}{1+e^{-x}} > 0$, such that a pixel-wise L2 or cross-entropy loss can be implemented.

In chapter 4, several other architecture decisions are discussed and evaluated. It turns out that the performance remains quite robust with respect to reasonable design choices (filter size, filter depth, number of layers, image size, etc.). Deeper CNNs like a full pretrained VGG16 encoder and transposed VGG16 decoder produce very detailed reconstructions, but the stored information moves from the latent space to the network weights. In theory, with enough encoder/decoder capacity, a single floating point could trigger the correct reconstruction. This loss of latent information can be observed by looking at the KL-divergence term: If the KL-divergence loss is near zero, it means that the reconstruction quality of training data presumably relies less on the relation between latent variables but more on the decoder's



Fig. 3.6: T-LESS object 5 recording; left: Primesense original, middle: 3D reconstruction from Primesense, right: Canon camera scene crop

memory. Since only the latent space is used for orientation estimation, more capacity is not necessarily beneficial.

3.3.2 Training Overview

During training, the input and desired output of the proposed Variational Autoencoder are fixed-size, square RGB images of an object, recorded from different viewpoints. Without loss of generality, RGB-D or only depth data can also be used. Their suitability will be discussed in the evaluation.

In the T-LESS [23] dataset 1296 centered object images covering the full view-sphere at constant radius are provided. In order to produce samples at all possible 3D orientations, each of these images has to be additionally rotated in-plane. This data is sufficient to successfully train the VAE framework. But unfortunately, this data is also never available in many robotic applications. Therefore, it is necessary to give up the dependency on pose annotated real sensor data.

One of the most relevant results of this thesis is the fact that a VAE can also be trained on purely synthetic data. In this context, synthetic data means to utilize virtual recordings from a textured 3D reconstruction. The 3D reconstruction can be generated by a hand-held RGB-D sensor which is swiped around the object. Through correspondences between the frames, an accurate 3D model can be synthesized (e.g. [43]). In T-LESS the textured 3D model is produced using AprilTags [44] to determine the camera pose. See Figure 3.6 for a comparison of the types of data. The 3D reconstruction loses some details and the edges are not straight compared to the original Primesense recordings, but from the viewpoint of a human, the pose is clearly recognizable.

3D reconstruction models obviate the need for pose annotations, serve as an unlimited data source of synthetic views and even allow the generation of arbitrary, realistic lighting condi-

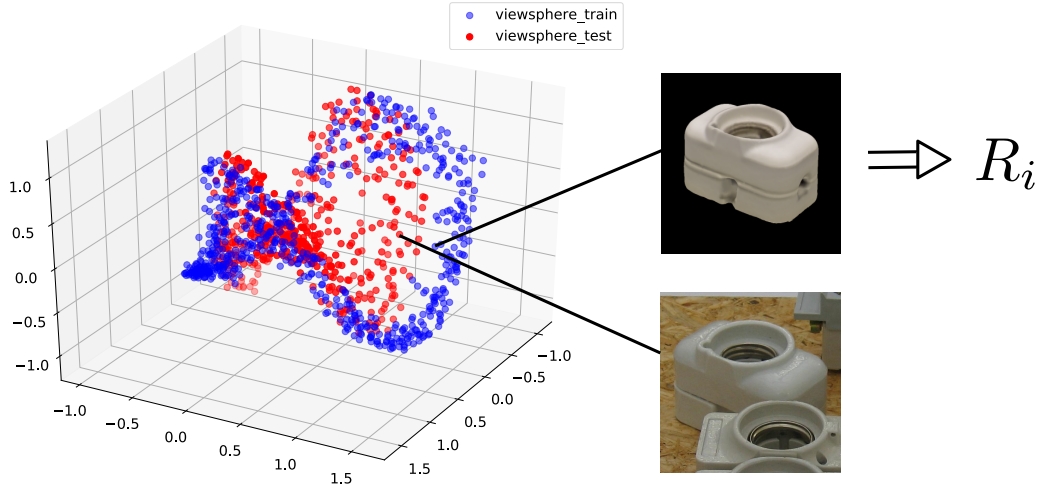


Fig. 3.7: Latent space of synthetic and real views after successful training

tions. The latter is advantageous because it reduces the risk of over-fitting to fixed lighting conditions which are present in many datasets.

For all these (virtual/real) views, labels in the form of rotation matrices are available. They describe the 3D rotational transformation from a camera to a 3D model coordinate frame. Instead of utilizing the labels directly in a regression or classification setting and suffering from all the issues discussed in section 3.1 related to representation and symmetries, the VAE learns descriptors unsupervised and solely based on the appearance of the object from different views.

Only after training, the rotation matrices from thousands of equidistantly sampled training views are assigned to their latent descriptors. The underlying idea is that, at test time, the encoder should be capable of projecting a scene crop containing the object to a meaningful descriptor, according to the object’s appearance which is closely connected to its orientation. Determining the nearest neighbors and their assigned orientations within the embedding of training views should therefore yield good estimates of the actual 3D orientation. This implies that the encoder does still recognize the view-specific appearance of the considered object in the test data.

Figure 3.7 shows an example of the three principle components of a training latent space from synthetic images and a test latent space from RGB scene crops. Although the spaces do not seem to be completely aligned, they are actually highly correlated. That means that the latent vectors of corresponding train and test orientations have very similar directions. This will motivate the use of cosine similarity as a **kNN** metric.

The decoder is no longer needed for the generation of pose hypotheses at test time but the crispness, segmentation and orientation of the reconstructions can give hints on how well the VAE manages to interpret the test image.

3.3.3 Augmented Training Procedure

So far, we naturally assumed that the VAE will produce embeddings according to the orientation of the object in training and test images. However, in unsupervised learning this assumption is actually quite naive, because there is usually much more possible variation in the images than only the orientation of the object. Even when training and testing with the same sensor, lighting conditions and background will change and foreground occlusions might appear.

For this reason, the next part describes a novel approach to steer the VAE embedding to become invariant against irrelevant variations through data augmentations at the input side. Simultaneously, the risk of over-fitting to sensitive features or noise that only exists in the training data is reduced due to larger variety in the training data. The final goal is, to make the encoder interpret the difference between real and synthetic data as just another irrelevant variation.

The idea generalizes the method of the denoising Autoencoder [58] (Figure 3.3b) which proposed to apply artificial noise on input images while computing the reconstruction loss on clean images. This way, the Autoencoder learns how to denoise images. But what does it mean in terms of encoding and decoding? It implicitly tells the encoder not to distinguish the input images by the type or strength of noise but rather to create very similar codes for an input image perturbed with any kind of noise. If the encoder becomes invariant to noise the decoder has an easier job to reconstruct the clean input, thus reducing the reconstruction loss.

In order to perform a successful template-based 3D orientation estimation, the object in the input image is randomly translated and scaled while the desired output stays unscaled and centered. As with the noise invariance, this should prevent the encoder to embed any 3D translational information into the latent variables so that the focus automatically shifts to encode the appearance with respect to orientation. Furthermore, these augmentations are important to account for uncertainties stemming from inaccurate object detections.

Figure 3.8 shows exemplary the impact of scale augmentation on the VAE robustness. A VAE is trained on binary object masks where only the azimuth is varied along one revolution. Elevation and cyclo angles stay fixed and the dimension of the latent space is $z \in \mathcal{R}^2$. Now, the VAE is trained with object masks at the original $scale = 1.0$ as input and desired output. From the trained model a test embeddings is created by inferring 36 masks at $scale = 1.0$ and at $scale = 0.7$. In the top left Figure 3.8, the resulting latent dimensions are depicted. It can be clearly seen that the embeddings are not invariant to scale which is expected since the VAE has only seen masks at $scale = 1.0$ during training.

Another VAE was then trained with inputs at uniformly distributed $scales \sim \mathcal{U}(0.7, 1.0)$ while the desired output was still fixed at $scale = 1.0$. In the bottom left Figure 3.8, the invariance against different scales for the same testing procedure can now be observed. Furthermore,

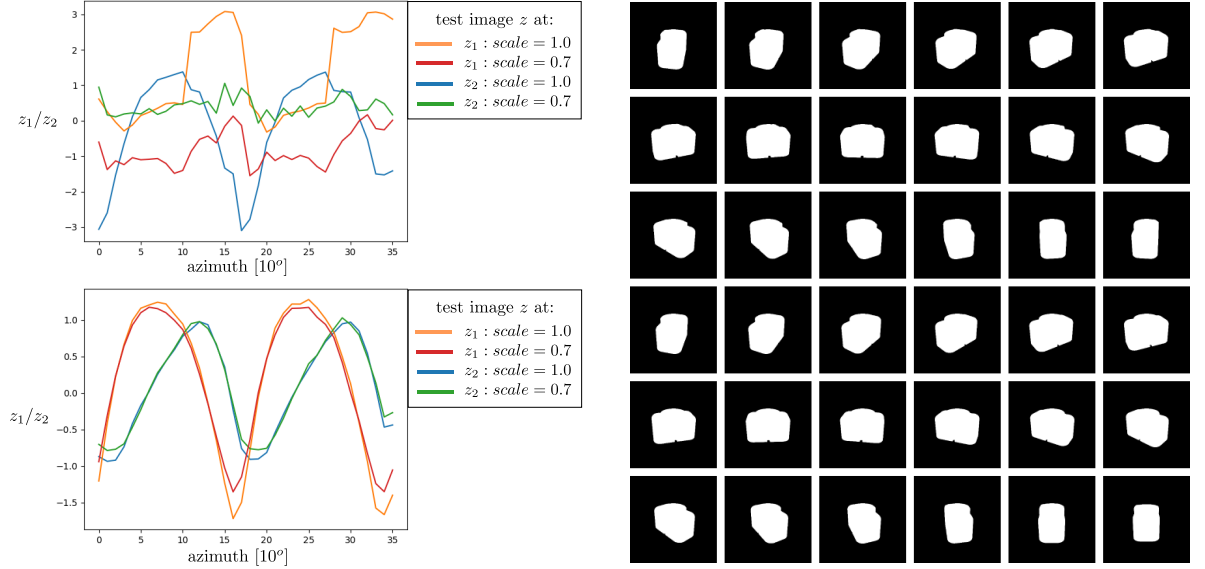


Fig. 3.8: Impact of scale augmentation on embedding; right: 36 binary masks at $scale = 0.7$; top left: VAE $z \in \mathcal{R}^2$ trained at $scale = 1.0$; bottom left: VAE $z \in \mathcal{R}^2$ trained at $scale \sim \mathcal{U}(0.7, 1.0)$

the test embedding at $scale = 1.0$ becomes much smoother and, remarkably, the two latent dimensions start to occur like sine and cosine functions on two full circular revolutions (4π)! Two revolutions, because the considered object has a symmetry plane which was inherently regarded. These observations justify the interpretation that the one-sided scale augmentation teaches the encoder to ignore scale differences and thus restricts the encoding to the rotational variations in the object mask appearances.

Using a good object instance segmentation algorithm, e.g. Mask-RCNN [15], it would be feasible to pipe a segmentation output into the trained VAE to generate a latent code. The nearest-neighbors within the training mask embeddings could then give hints about the orientation. The training masks could for example be generated from a CAD model. However, this method would omit a lot of useful information in the texture and internal shape of the object while not being end-to-end. To apply the VAE directly on RGB scene crops, much more augmentation is necessary.

The encoder has to become invariant against any kind of background clutter. If the training happens only on object views with black background, as shown in Figure 3.9 (left), the encoder will fail to distinguish the object from background at test time. Therefore, a random image background from some dataset as, for example, VOC2012 [9] is introduced at the input. Since the desired output background remains black, the VAE learns to internally segment the object and ignore the background.

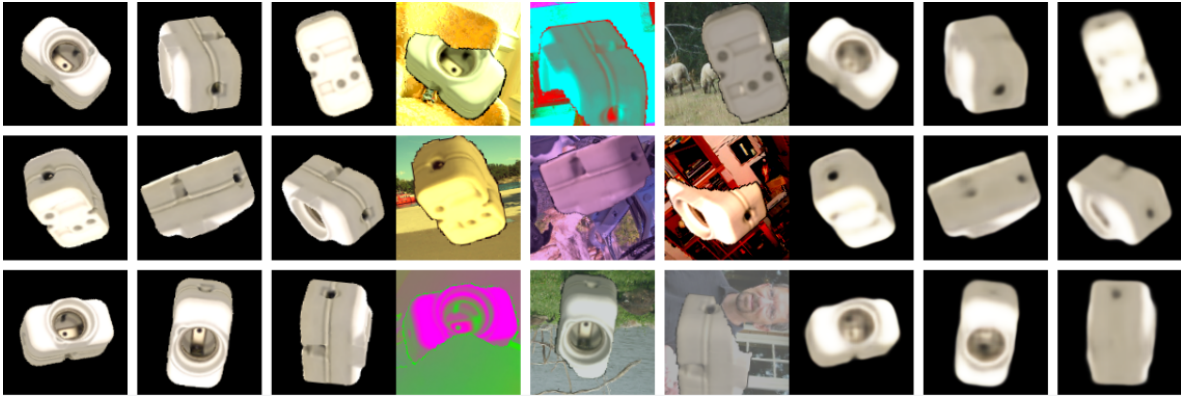


Fig. 3.9: Left: Clean virtual view of textured 3D model of object 5 (desired output); middle: augmented input; right: decoder reconstruction after 35000 iterations

Sensitivity to occlusions is often a point of criticism against template-based approaches. To make the encoder robust against occlusions, randomly translated object masks are used to realistically cut out pieces from the input object view. In practice, this process is performed repeatedly until a certain amount (e.g. 0 – 25%) is occluded. Consequently, the VAE must learn how to reconstruct missing parts of the input and the path of least resistance is to make the encoding invariant to occlusions. This augmentation is supposed to avoid over-fitting to few features that might be occluded during testing. On the other hand, too much artificial occlusion complicates the disentanglement during training.

Next, another advantage of training with synthetic views from 3D reconstructions is exploited. To avoid influences of specific lighting conditions on the latent space, we generate light sources at random positions using the Phong lighting model [46] to illuminate the 3D reconstruction. The desired output views are only illuminated using ambient light. This way, invariance against various kinds of lighting conditions is achieved.¹

Finally, several other image augmentations are used by chance:

- Gaussian blur
- Additive Gaussian Noise
- Add
- Multiply
- Contrast normalization

These augmentations occur at random strength and sometimes act on single color channels (except blurring).

¹Note, during the 3D reconstruction the lighting conditions should be close to ambient to prevent an initial brightness bias coded into the vertex colors. Otherwise, this issue can be resolved by clamping the vertex color values in the HSV color space or extracting an albedo map

Figure 3.9 depicts an example of clean desired output, final augmented input and decoder reconstruction from a mini-batch. It can be observed how the VAE manages to revert all of the described augmentations on the synthetic training views.

A property of Autoencoders is that they produce slightly blurry outputs. The blurriness originates from the pixel-wise evaluated loss function which assumes conditional independence between the pixels and does not permit translational variance at pixel level. Other approaches like 'Autencoding beyond pixels' [33] or 'Adversarial Autoencoders' [40] propose to combine the VAE with Generative Adversarial Networks (GANs) [13] where the VAE acts as the generator. A discriminator tries to tell the inputs of the VAE apart from the VAE reconstructions. In an adversarial training scheme, the VAE improves its capability to produce sharp, realistic outputs. Therefore, using GANs is beneficial in many generative settings, although being harder to train. However, reconstruction crispness is not necessarily linked to the information embedded into the latent space. It can also simply mean that the decoder is more capacitive. Furthermore, the pixel-level accuracy requirement can be crucial for accurate orientation estimation. Nevertheless, future experiments should be conducted in this direction.

The 3D model reconstruction sometimes lacks details of the real object so that an over-fitting on its features should be avoided. However, strong augmentations minimize the risk of over-fitting so that early stopping of the training is usually unnecessary. A drawback of unsupervised, appearance-based methods compared to pose-supervised methods is that minor details, like the small notch of object 5 (Figure 3.6), which break the symmetry have only little influence on the total loss and are thus sometimes ignored.

3.3.4 Training embedding generation and testing procedure

The training conditions and parameters are further described in chapter 4. This section explains the utilization of the trained VAE for 3D orientation estimation. Figure 3.10 illustrates the whole procedure. After the training phase, object orientations should produce meaningful latent variables, i.e. close orientations with similar appearance should produce similar latent codes. Now, to utilize this property for 3D orientation estimation, first, a training embedding is produced:

1. Densely sample **clean** (synthetic) templates from a view-sphere based on a refined icosahedron [17]
2. Rotate each template in-plane at fixed intervals and assign the final $\{R_{cam2obj}\}$ rotation matrices, examples can be seen in Figure 3.9 (left)
3. Generate and assign latent codes by feeding the templates batch-wise into the trained encoder

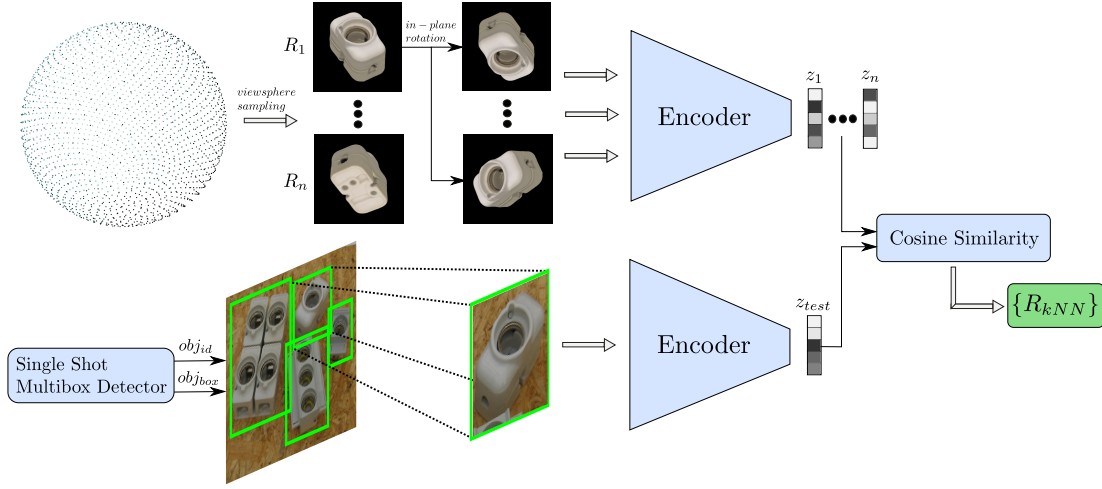


Fig. 3.10: VAE orientation estimation; top: offline sampling and generation of training embedding; bottom: online inference of scene crops; right: nearest neighbor calculation

The test procedure is summarized in pseudo-code 1. First, the Single Shot Multibox Detector (SSD) produces one or more object crops from a scene which are resized and fed into the encoder. Each resulting test latent code \mathbf{y} is compared with the training embedding codes \mathbf{x} through cosine similarity:

$$\cos(\mathbf{x}, \mathbf{y}) = \frac{\mathbf{x} \cdot \mathbf{y}}{\|\mathbf{x}\| \cdot \|\mathbf{y}\|} \quad (3.14)$$

The highest similarities are determined in a [kNN](#) search and the corresponding rotation matrices are returned:

Algorithm 1 Joint 2D object and 3D orientation detection

Data: scene_im, latent_space, R_all, topk

Result: obj_id, obj_box, R_kNN

obj_id, obj_box \leftarrow SSD(scene_im)

obj_crop \leftarrow squaredResizedObjCrop(scene_im, obj_box)

z_test \leftarrow encoder(obj_crop)

n \leftarrow len(latent_space)

for i = 0 **to** n - 1 **do**

 z_i \leftarrow latent_space(i)

 cosine_sim(i) \leftarrow $\frac{z_i \cdot z_{test}}{\|z_i\| \|z_{test}\|}$

end

sorted_sim_idcs \leftarrow argsort(cosine_sim)

for k = 0 **to** topk - 1 **do**

 R_kNN(k) \leftarrow R_all(sorted_sim_idcs(end - k))

end

return obj_id, obj_box, R_kNN

3.4 Multiple Objects

Until now, we have considered to train a VAE only on one object. Simply training the VAE with multiple objects can result in ambiguous latent variables where the objects might not be clearly separable at certain views or occlusions. This would unnecessarily disturb the orientation estimation since the object class is already predicted from the object detection.

A simple solution is to individually train several VAEs on different objects and load the weights of all encoders into the GPU memory. Obviously this approach is limited by the total memory size, but in practice the above stated encoder architecture consists only of 1.89 million weights which corresponds to a memory size of around 7.2MB. Additionally, depending on the sampling intervals and latent space dimension, the precomputed training embedding needs 5 to 50MB memory space. Since modern GPUs have a memory size of ~ 12 GB, we could theoretically load the weights and training embeddings of numerous (>200) objects. Despite some practical overhead and memory demands of the detector, this allows concurrent 2D object detection and orientation estimation of many objects in parallel. Since the object detection yields the object class, we can then pick the corresponding encoder for inference.

Training several VAEs from scratch is time-consuming. If there was a way to infer the object class from the detector, we could jointly train one VAE on several objects without inter-class confusion. Luckily, this can be done in a very simple manner introduced by Sohn et al. [54]. The [Conditional Variational Autoencoder \(CVAE\)](#) simply concatenates a one-hot encoding of the input class (object) with the latent code produced by the encoder. This additional data informs the decoder of the object class that should be reconstructed, thus facilitating the training. The model is only slightly changed at the fully-connected layer of the decoder, which now maps a latent code of size $z \in \mathcal{R}^{n+c}$, with n dimensions of the original latent space and c number of considered objects, to the first stack of feature layers. The testing procedure is quite similar:

1. Use [CVAE](#) encoder to create multi-view training embedding from each synthetic object model
2. Use [CVAE](#) encoder to create a latent code from one or more scene crops
3. Maximize cosine similarity only against the multi-view embedding of the corresponding object

Through the exploitation of the induced object labels during training and testing, a single CVAE model has the ability to estimate the orientations of several objects. As shown in the next chapter, drawbacks include an increased training time and a slightly reduced accuracy. Hypothetically, the decoder might lack the capacity to jointly learn the multi-view reconstruction of multiple objects. On the other hand, more capacity would again shift the information from the latent space into the decoder as described before.

In conclusion, both approaches present a practical solution for multi-object, multi-view orientation estimation. While training several single-object VAEs gradually increases the memory demands, it is definitely feasible on modern GPUs. In both cases, a separate training embedding for each object is produced from which the nearest neighbors are determined.

3.5 Inference Time

In robotic applications short inference time is desirable, especially in dynamic environments. Short run-times enable repeated predictions and thus increase the confidence and accuracy compared to single pose estimates. Furthermore, tracking algorithms can be used if a frequent stream of predictions is available.

The computational demands at test time are defined by the 2D detection, encoder inference, computation of the cosine similarity and the nearest neighbor search.

The cosine similarity operation basically consists of a high-dimensional matrix-vector multiplication and normalizations. Computing the cosine similarity from the test code to all codes in the training embedding therefore has complexity $\mathcal{O}(nm)$ with n dimensions of the latent space and m latent codes in the training embedding. As this can be parallelized well, a GPU computation is favorable.

The nearest neighbor is simply the maximum entry in the resulting vector with complexity $\mathcal{O}(m)$. In our experiments, this memory lookup works faster on modern CPUs using Numpy than on modern GPUs using Tensorflow.

At a common configuration with $n = 64$ and $m = 92232$, an averaged benchmark of the individual parts of the pipeline is depicted in Table 3.1.

Table 3.1: Inference time of pipeline blocks

	8 CPUs	GPU
SSD	-	$\sim 17\text{ms}$
Encoder	-	$\sim 6\text{ms}$
Cosine Similarity	2.5ms	1.3ms
Nearest Neighbor	0.3ms	3.2ms
		$\sim 24\text{ms}$

We conclude that the whole system is real-time capable at $\approx 42\text{Hz}$. This enables many robotic applications and leaves room for tracking algorithms.

4 Experiments and Discussion

In the following, different variants of the [Autoencoder \(AE\)](#) and the Single Shot Multibox Detector (SSD) are evaluated. The focus lies on the main contributions of this thesis, i.e. an ablation study of the augmentations related to the performance of the VAE.

As indicated, the experiments are mainly performed on the challenging T-LESS dataset which is characterized by the obstacles described in the previous chapter 3. Other popular datasets, as the LineMOD dataset [20] exhibit less symmetries and more distinctive shapes and textures, thus are easier to solve. The reason for the popularity of the LineMOD dataset might also stem from a facilitated pose evaluation of unambiguous objects.

4.1 Evaluation Metrics

Before presenting the results, the evaluation metrics for 3D orientation estimation and 2D detection are defined.

4.1.1 Axis-angle Rotation Error

If ambiguities are non-existent or resolved, a single-valued absolute rotation error can be computed between the estimated rotation matrix $\mathbf{R}_{est_cam2obj}$ and the ground truth rotation matrix $\mathbf{R}_{gt_cam2obj}$. The rotation error in matrix notation is defined as

$$\mathbf{R}_{err} = \mathbf{R}_{est_cam2obj}^T \mathbf{R}_{gt_cam2obj} \quad (4.1)$$

\mathbf{R}_{err} can be converted into an axis-angle representation such that the absolute rotation error $e_R \in [0^\circ, 180^\circ]$ can be described as the magnitude of the angle:

$$e_R = \arccos \left(\text{Tr}(\mathbf{R}_{err} - \mathbf{I}) / 2 \right) \quad (4.2)$$

Using this metric an intuitive estimation of the object orientation error is established. It is convenient in scenarios where the exact orientation matters and is uniquely defined. However, for the evaluation of ambiguous objects, it is necessary to tediously predefine the symmetries to correct the acquired rotation errors.

4.1.2 Visible Surface Discrepancy

It is possible to abstain from this necessity by utilizing an ambiguity-invariant pose error function: The [Visible Surface Discrepancy \(VSD\)](#) [22] measures the deviation between the visible 3D surface of a rendered object model at ground truth pose V and at the estimated pose \hat{V} . For each object depth pixel p a matching cost c is computed which linearly increases with the depth distance d until reaching a threshold τ . For $d \geq \tau$ the pixel matching cost is clipped.

$$c(p, d, \tau) = \begin{cases} d/\tau & \text{if } p \in \hat{V} \cap V \wedge d < \tau \\ 1 & \text{otherwise for } p \in \hat{V} \cup V \end{cases} \quad (4.3)$$

The total matching error e_{vsd} is simply the average over the pixel-wise costs.

$$e_{vsd} = \frac{1}{N} \sum_{p \in \hat{V} \cup V} c(p, d, \tau) \quad (4.4)$$

where N is the number of pixels in $\hat{V} \cup V$. The visible surface discrepancy measures the difference in appearance of the estimated and ground truth object views rather than measuring a pose error from some $SO(3)$ representation. Thus, the issue of interpreting pose estimations resulting from symmetric views and/or objects is circumvented. The [VSD](#) metric is actually quite similar to the reconstruction objective of the [VAE](#) which also depends on pixel-wise distances. A variant of the [VSD](#) metric, where the per-pixel cost c is zero up to the distance threshold τ , will be the official metric for the SIXD challenge [21] and, for the sake of comparability, also used here. Furthermore, only views with $\frac{\text{visible object pixels}}{\text{all object pixels}} > 10\%$ are considered.

Both metrics can be a suitable quality measure in different settings, therefore, both are evaluated in this thesis. To individually evaluate the VAE, only the 3D orientation estimation is regarded, i.e. translation is not estimated. Later the whole pipeline is evaluated.

4.1.3 Mean Average Precision

Object detectors are commonly evaluated in terms of [mean Average Precision \(mAP\)](#) [10]. Precision is defined for each class as the fraction of correct class predictions for predicted boxes with an intersection over union (iou) ≥ 0.5 with the ground truth box. Average Precision refers to the precision at a set of 11 different recall levels $r \in \{0, 0.1, \dots, 1\}$, where recall is defined as the number of correctly predicted bounding boxes from the total number of existent ground truth bounding boxes of that class. To obtain the recall levels we vary a softmax prediction

threshold and interpolate the generated {precision (p), recall (r)} pairs. The Average Precision (AP) is computed as

$$AP = \frac{1}{11} \sum_{r \in \{0, 0.1, \dots, 1\}} p_{interp}(r) \quad (4.5)$$

Finally, **mAP** simply relates to the mean AP over all classes.

4.2 Single Shot Multibox Detector

SSD [36] is an essential part in the presented pose estimation pipeline as it quickly and reliably detects all objects in an RGB image. Yet, it is not the focus of this thesis to thoroughly evaluate the object detector since a tremendous amount of research is already devoted to it. For more details, I refer to [24].

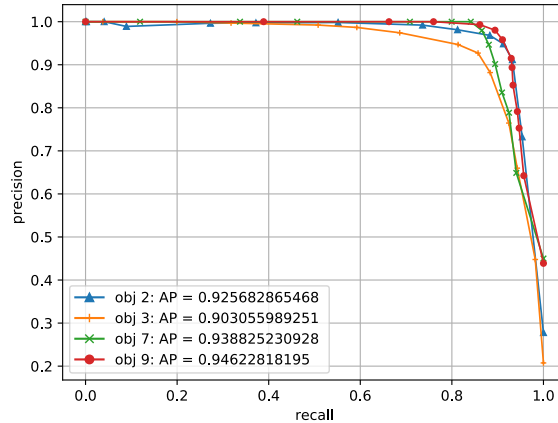
I trained SSD on all 30 objects that appear in the canon scene recordings of the T-LESS dataset using their bounding box annotations. To evaluate the performance, the 20 scenes - each containing 504 views - are split into 19 training scenes and 1 test scene. The following evaluation only considers objects 2,3,7,9 in test scene 12.

The work is based on the Tensorflow implementation of [45]. The starting point is a VGG16 base network pretrained on ImageNet and a randomly (Xavier[12]) initialized head network which was modified to match the 30 classes + 1 background class. The hyper-parameters and training conditions are summarized in table 4.1

Table 4.1: SSD characteristics

Dataset:	T-LESS	Optimizer:	Adam
Input shape:	300 x 300	Learning rate:	1.0E-04
Batch size:	22	Training Time (DD:HH:MM):	02:03:36
Inference Time / Freq:	17 ms / 59 Hz	Iterations / Epochs:	267 595 / 614

Even though some of the T-LESS objects are hard to distinguish, the object detection task on 30 different rigid instances is fairly simple compared to the detection of articulated, deformable instances from broader classes. Furthermore, training and test data can be assumed to origin from very similar distributions (light conditions, background, etc.). Unsurprisingly, the results are very accurate with an $mAP = 0.9284$ on the test set, see Figure 4.1. Note, that only slightly worse results can be achieved after a fraction of the total training time of 2 days on a Geforce 1080 GTX.



(a) Object detections in Primesense test scene 12 (b) Average-precision for objects $\{2,3,7,9\}$ in scene 12

Fig. 4.1: Single Shot Multibox Detector; a) example scene; b) precision-recall statistics

The training images are whitened and augmented through random cropping, resolution changes, horizontal flips, shape and color distortions. The bounding boxes are adjusted accordingly. The authors of [36] report strong improvements through these augmentations. They increase the variety of training data and avoid over-fitting. In combination with pre-trained weights, a small experiment verified that a subset of only 1000 annotated scene images is sufficient to train the SSD. The labeling of bounding boxes is much less cumbersome than creating pose annotations, therefore supervised learning for detection is much more feasible in real world scenarios.

However, a future goal is to train the object detection on synthetic views of textured 3D models, too. First attempts of training with randomly rendered models on the background of the VOC dataset [9] did not yield satisfactory results on particularly smaller objects. Table 4.2 reports the Average Precision (AP) values per object on real and synthetic data. Also refer to Figure A.1 in the appendix.

Table 4.2: Average Precision SSD

AP scene 12	object 2	object 3	object 7	object 9	mAP
real Canon scenes	0.926	0.903	0.939	0.946	0.928
synthetic with VOC bg	0.166	0.154	0.298	0.584	0.301

4.3 VAE Training Conditions

Before presenting the results of the VAE, the training conditions and variations are shortly outlined. The model architecture, loss functions and the types of augmentation have already been discussed. Not every design decisions can be scientifically grounded due to the sheer mass of possible modifications. However, to train the VAE on orientation estimation is comparatively robust with respect to different choices of these parameters, unlike for example Generative Adversarial Networks (GANs).

The training is performed on $128 \times 128 \times 3$ RGB images as it seems enough to cover the details of texture-less objects. Larger inputs increase the training time but do not stimulate the performance. The batch-size is 64, chosen to fill the GPU memory. For the gradient-based optimization I chose Adam [27] which includes lower-order moments and has good convergence properties. It does not require intensive parameter tuning, thus the learning rate can be fixed at $1e-4$. The implementation is done in the graph-based library Tensorflow and the computations are executed on a GeForce Titan X GPU.

4.4 VAE Evaluation Results on T-LESS

For better comparison between the VAE variants, the input crops are generated using ground truth bounding boxes. Later, the whole pipeline including object detection and refinement is examined. Due to computational limitations, all possible variants are first validated on the fifth object from the T-LESS dataset. Finally, the generality of the results is proven on other objects. Generality is also proven with respect to different kinds of sensors. The VSD error is analyzed on the Kinect and Primesense recordings of different scenes and the total axis-angle rotation error is evaluated on Kinect, Primesense and Canon RGB scene images.

Chronologically, the first successful experiments were achieved on binary object masks from CAD models. This motivated the training on real RGB images and finally the training on synthetic views of a 3D model reconstruction. As a baseline for the synthetic data, the training on 1296 Canon recordings \times 36 artificial in-plane rotations is first considered.

4.4.1 Training on Canon images

For training and testing, object crops from Canon images are utilized here. The T-LESS dataset offers images of isolated object on black background which is replaced by random VOC images during training. Additionally, all augmentations described in Section 3.3.3 are applied except for the dynamic light sources for obvious reasons.

For better intuition, the axis-angle error of the 3D rotation is first depicted in a histogram which describes the distribution of errors within the 504 views from a half-sphere around scene

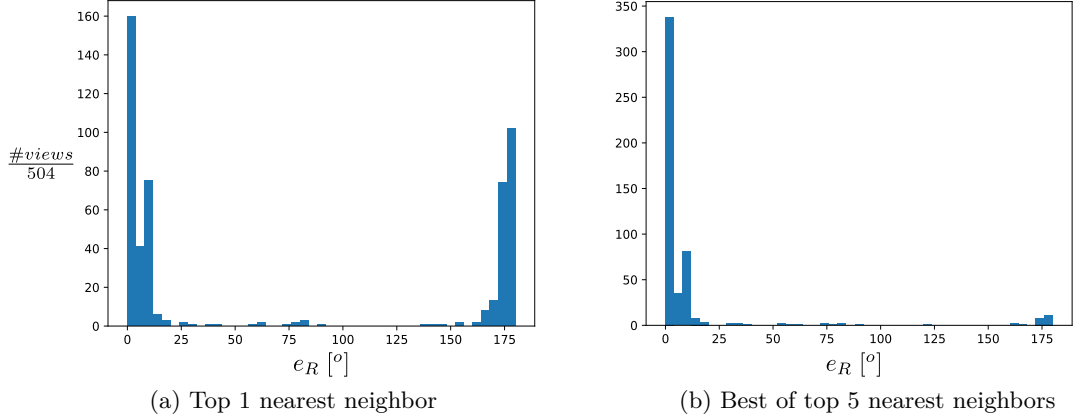
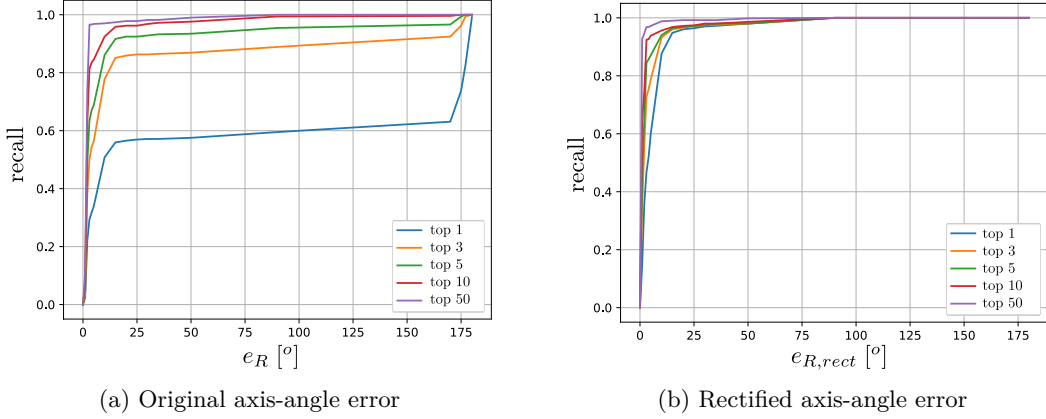


Fig. 4.2: Axis-angle error histogram: object 5, Canon scene 2, 4° intervals

Fig. 4.3: Interpolated cumulative error-recall curves for Top $k \in \{1, 3, 5, 10, 50\}$ NNs

2, Figure 4.2a. It can be observed how the orientation of the majority of views is either quite accurately estimated or with an absolute rotation error close to 180° . This comes from the fact that object 5 is actually an almost symmetric object (Figure 3.6) where only the (sometimes hidden) small notch distinguishes a 0° error view from a 180° error view. Taking into account this ambiguity, we can assure ourselves that the 3D orientation is reliably predicted among the top 5 nearest neighbors, see Figure 4.2b.

How can the average performance of the VAE be condensed into a single scalar? The average or median angle error is rather meaningless due to the ambiguity. For this particular object 5 one could rectify the average error by taking the distance to either 0° or 180° . This method yields a nearest neighbor average error of 6.48° . In order to represent the whole distribution in a single number we can describe the performance as the area under the e_R / recall curve (AUC_{re}). The recall curves for object 5 are depicted in Figure 4.3a with the original axis-angle errors and in Figure 4.3b with the rectified axis-angle errors which neglects the notch and

treats object 5 as being symmetric. The corresponding area under the top 1 error distribution is computed as

$$AUC_{re} = \frac{1}{90} \int_0^{90} recall(e_R) de_R \quad AUC_{re,rect} = \frac{1}{90} \int_0^{90} recall(e_{R,rect}) de_{R,rect} \quad (4.6)$$

The e_{vsd} error can only be computed for the RGB-D sensors. The area under the e_{vsd} / recall curve reads:

$$AUC_{vsd} = \int_0^1 recall(e_{vsd}) de_{vsd} \quad (4.7)$$

Table 4.3 shows how the Canon-based training generalizes to Kinect and Primesense RGB recordings without a strong loss in accuracy.

Table 4.3: AUCs for VAE training with Canon images

Test RGB	AUC _{re}	AUC _{re,rect}	AUC _{vsd}
Canon	0.571	0.916	-
Primesense	0.483	0.886	0.890
Kinect	0.517	0.890	0.917

The 3D orientations of fully visible or moderately occluded object views are reliably and accurately predicted if neglecting the ambiguity. Mainly views with very strong occlusions are wrongly assigned. The influence of occlusion will be explored in the succeeding sections.

4.4.2 Training on synthetic views

The generalization from Canon to Kinect and Primesense sensors supports the assumption that training on synthetic data is also possible. Nevertheless, the differences between real and synthetic images must not be underestimated. Only a preprocessing pipeline of various strong augmentations that can be inverted by the VAE leads to similar accuracies as training with real data. In the following, an empirical ablation study of the different augmentations and its implications for the latent codes is presented.

Unless otherwise stated, the following networks are trained for 25000 iterations which, depending on the number of augmentations, takes about 4-6 hours on a single GPU. During training the networks are constantly evaluated on their AUC_{vsd} .

Table 4.6 describes the impact of individual color augmentations on accuracy described in AUC_{vsd} . In order to account for the randomness of the predictions, the performance of the last three checkpoints between iteration 20000 and 25000 are averaged and the standard deviation is given in brackets. To produce the training embedding, we take 2562 synthetic recordings of the object from an equidistantly sampled sphere at constant radius (650mm).

Each view is then rotated in-plane at 36 different angles to produce a total of 92232 latent codes with latent size 16 that represent the whole $SO(3)$.

For comparison, color (Table 4.4) and shape (Table 4.5) augmentation strengths are within a fixed range during these experiments. To prevent any over-fitting on the augmentations, they are applied uniformly at random. Scale and translation strengths are given w.r.t. image size. The random background substitutions are not mentioned, although they are absolutely crucial as otherwise the encoder does not learn to distinguish the object from the background. Therefore, this modification is part of every experiment. The background undergoes the same color augmentations as the foreground object.

Table 4.4: Input color augmentation strengths of experiments in 4.6

	dyn. light	add	contrast	multiply	invert
30% chance (10% per channel)	amb:[0.8,1.0] diff: [0.6,0.8]	[-10,10]	[0.5,2.0]	[0.7,1.3]	per channel

Table 4.5: Input shape augmentation strengths of experiments in 4.6

	occlusion	scale	translation (x, y)
100% chance	[0,0.25]	[0.9,1.15]	[-0.1,0.1]

Table 4.6: Ablation study on color augmentations

Test RGB	dyn. light	add	contrast	multiply	invert	AUC _{vsd}
Primesense	✓					0.472 (± 0.013)
	✓	✓				0.611 (± 0.030)
	✓	✓	✓			0.825 (± 0.015)
	✓	✓	✓	✓		0.876 (± 0.019)
	✓	✓	✓	✓	✓	0.877 (± 0.005)
		✓	✓	✓		0.861 (± 0.014)
Kinect	✓					0.461 (± 0.022)
	✓	✓				0.580 (± 0.014)
	✓	✓	✓			0.701 (± 0.046)
	✓	✓	✓	✓		0.855 (± 0.016)
	✓	✓	✓	✓	✓	0.897 (± 0.008)
		✓	✓	✓		0.903 (± 0.016)

Strictly speaking, all combinations need to be examined in order to estimate mutual dependencies. Since it is practically infeasible to train such a high number of networks, several combinations are presented to convey an intuition of the effects.

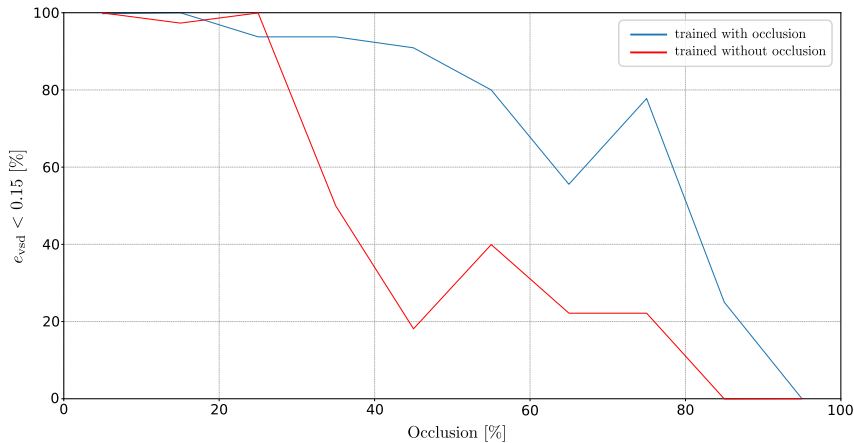


Fig. 4.4: VAE trained with and without occlusion augmentation

The experiments in Table 4.6 show the importance of various color augmentations for transferring the pose information from synthetic training data to real test data. In fact, the simple conclusion that more color augmentations produce better performance pretty much holds. Even inverting the color channels improves the AUC_{vsd} . This implies that, without color distortions, the network over-fits on color information which is present in the synthetic data but not in the real data. On the other hand, low-level information about edges and uniform intensity regions is preserved. These kind of features seem to be recognizable in both, real and synthetic data such that the reconstruction of the clean image remains feasible.

The simulated diffuse lighting conditions from random directions do not seem to produce clear improvements of the AUC_{vsd} . However, this is likely due to over-fitting to the similar vertical incidence of light which is present in both, the 3D model texture and test scene 2. Considering scene 3, where the lighting conditions are very different, this augmentation is crucial because here the lateral faces are instead brighter. If no artificial lighting is applied the VAE assumes bright faces are oriented upwards as in the model texture. The accuracy improvement from artificial dynamic light in scene 3 is thus significant. It shows that artificial lighting reduces the effect of light bias in the 3D reconstruction, making the estimator more robust to different lighting conditions. Stronger 2D color augmentations, as described below (Table 4.7), can also reduce the effect of light bias. After all, the importance of simulated light depends on the quality of the 3D reconstructed texture.

Next, the effect of random occlusions during training is examined. Figure 4.4 shows the amount of VSD errors below a threshold of 0.15 at different levels of object occlusions, divided into 10 bins. The threshold is a subjectively chosen bound for perceptual correctness of the prediction. It does illustrate that a VAE trained with occlusions in the range of 0–25% is far more robust against strong occlusions at test time than a VAE trained without occlusions.

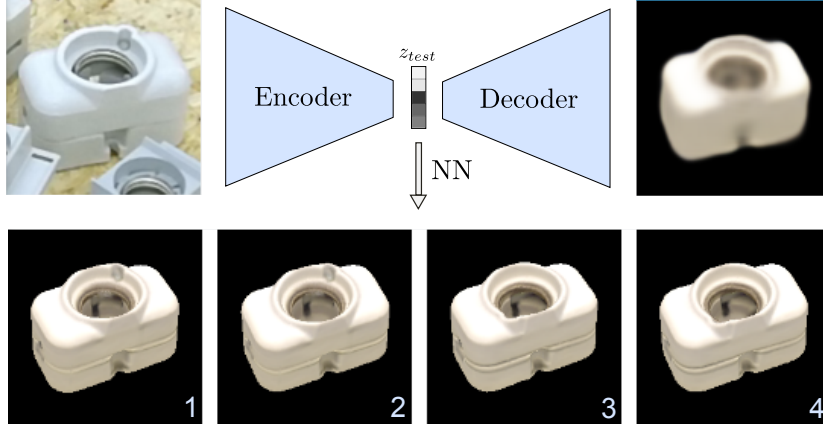


Fig. 4.5: Reconstruction of Kinect RGB crop of scene 11 and synthetic Nearest Neighbor views

Figure 4.5 depicts how the trained model generalizes to RGB test data from the Kinect that it has never seen before. The decoder reconstruction on the top right is the result of interpreting the low dimensional code from z . It clearly segments the object from the background and reconstructs the parts occluded by objects in the foreground.

The four nearest neighbors of the latent test code belong to two synthetic views with very similar orientation to the test object and two synthetic views that are rotated by $\sim 180^\circ$. It characterizes the learned representation in which the similarity of latent codes is directly connected to similarity in appearance.

The question naturally arises, whether even stronger augmentations will lead to better performance or failure. Table 4.7 and 4.8 show that stronger color distortions actually continue to improve the performance. Naturally, escalating augmentations even further is at risk to disguise important artifacts which results in lower performance.

Table 4.7: Stronger augmentation

	dyn. light	add	contrast	multiply	invert
50% chance (30% per channel)	amb:[0.8,1.0] diff: [0.6,0.8]	[-20,20]	[0.4,2.3]	[0.6,1.4]	per channel
	scale	translation	occlusion		
	[0.85,1.25]	[-0.15,0.15]	[0,0.25]		

Table 4.8: AUCs for VAE training with synthetic images

Test RGB	AUC _{re}	AUC _{re,rect}	AUC _{vsd}
Canon	0.519	0.911	-
Primesense	0.481	0.899	0.897
Kinect	0.513	0.919	0.911

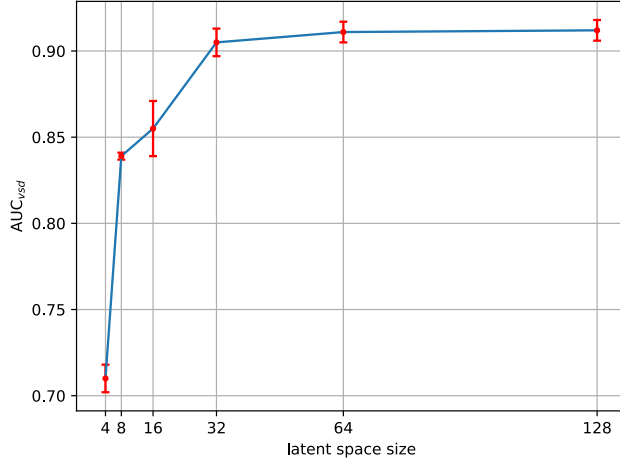


Fig. 4.6: Dependency of performance on latent space size (Kinect)
standard deviation in red

Through the heavy input augmentation, the performance of training with synthetic data (Table 4.8) is now actually on par with the training using real Canon data (Table 4.3). Constraining the unsupervised representation learning makes training with synthetic data feasible. This main result of this work allows fast and accurate single RGB frame pose estimation without the need for real data pose labels.

4.4.3 Latent Space Size

Until now, the latent space size has been fixed to $z \in R^{16}$. Theoretically, more dimensions should allow to store more information while taking more computation time and memory. Figure 4.6 shows how the AUC_{vsd} evolves over different latent space sizes. All augmentations from Table 4.4 are utilized, besides inversion. The performance first quickly rises with the latent space size and begins to saturate at $n = 64$.

Since the VAE is very fast and memory efficient compared to other orientation estimation approaches, it is reasonable to choose a higher latent space size in most applications. The performance of a $n = 128$ dimensional latent space combined with the strong augmentations from 4.7 lead to the best final accuracies, Table 4.9.

Table 4.9: Best VAE model with $z \in R^{128}$

Test RGB	AUC _{vsd}
Primesense	0.914 (± 0.003)
Kinect	0.931 (± 0.001)

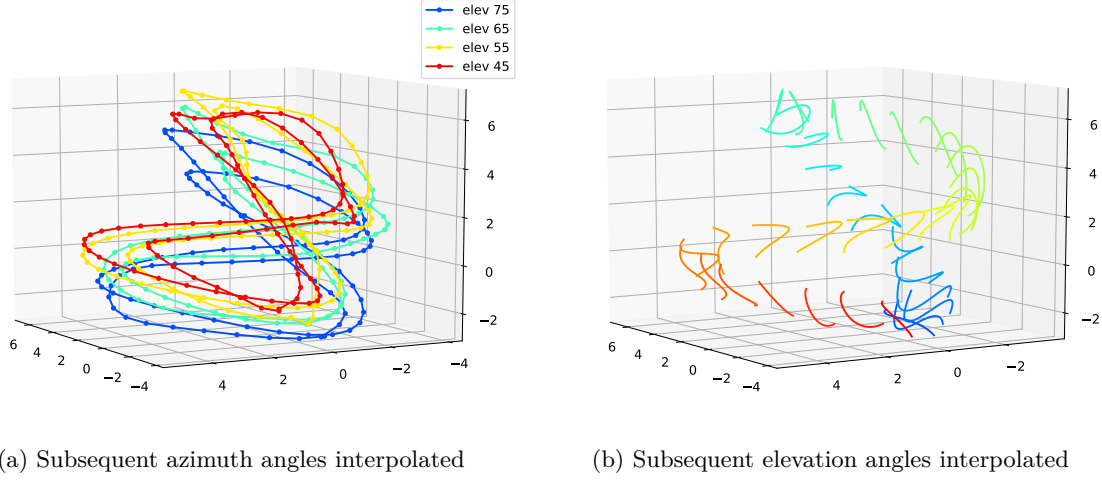


Fig. 4.7: 3 Principle Components of test latent space of scene 2, object 5

4.5 Analyzing the Latent Space

This section investigates the characteristics of the latent space which finally motivates the use of cosine similarity as a Nearest Neighbor (NN) metric.

Until now we have treated the latent space as a black-box. To gain a better understanding of the learned representations, the space is first visualized. Therefore, we run a [Principal Component Analysis \(PCA\)](#) with 3 components on the test latent codes to extract the directions with highest variance. The test latent codes are created from real sensor data of a test scene. The recordings are taken at 7 different elevation levels (10° intervals) and 72 different azimuth angles (5° intervals).

In Figure 4.7a each color depicts the latent codes from one elevation level. As in the toy example 3.8 with segmentation masks, we can observe that the nearly symmetric object 5 creates a path of two revolutions in the latent space. Figure 4.7b depicts the latent code motion when changing the elevation at fixed azimuth angles. The visualizations show how the VAE manages to arrange object orientations in the latent space.

During the experiments, it has been noticeable that the magnitude of the latent vectors is usually greater in the training set embedding than in the test set embedding. To analyze the effect of magnitude, we produce decoder reconstructions at several scales of a test latent vector, see Figure 4.8. Surprisingly, the 3D orientation of the object is hardly affected by the magnitude. It can be observed, how the reconstructions become crisper and richer with the length of the latent vector. An all-zero latent code produces only a blurry circle which might depict the mean reconstruction of all object orientations. At scale $s = 3.0$ the reconstruction seems unnaturally shiny and patchy.

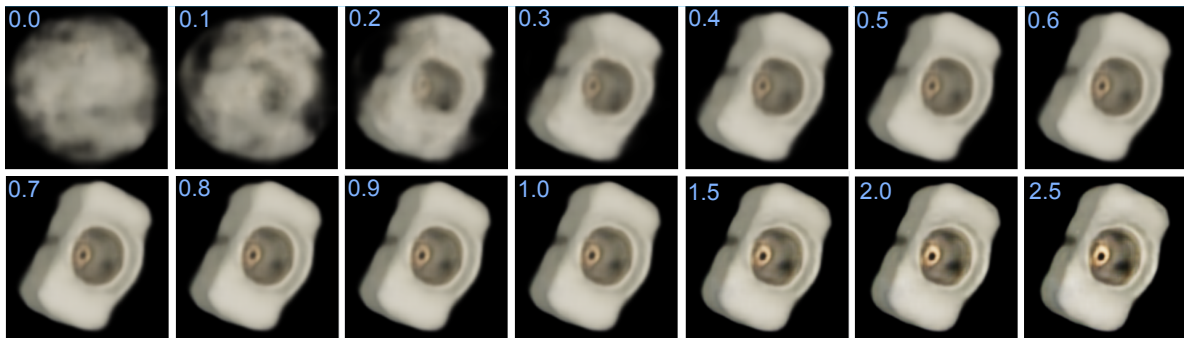


Fig. 4.8: Decoder reconstruction of a test latent vector scaled by factor $s \in [0, 2.5]$

Since the orientation embedded in the latent code is unaffected by its length, the use of the cosine similarity as a metric for Nearest Neighbor search is chosen. The cosine similarity computes the dot product between two normalized vectors. The result only depends on the angle between both vectors. Using the more common Euclidean distance produces worse results in all experiments.

The difference in magnitude decreases with the strength of augmentations being applied at training time. Augmentations have also been shown to improve the performance of the VAE. This raises the question whether the magnitude of the test latent vector is a qualitative characteristic for the orientation prediction. In fact, the magnitude ratio between a test latent vector and its nearest training latent vector correlates with the accuracy of the predicted orientation. Unfortunately, the correlation is not strong enough for a secure statement on the uncertainty of the prediction. However, the visualizations of the training and test embeddings still enable a good insight into the training performance.

4.6 Variational Autoencoder vs. plain Autoencoder

In the following, the reasons for and against Variational Inference with Autoencoders related to 3D orientation estimation are discussed. The assumptions are based on experiments and hypotheses about the data.

Variational Autoencoders have several theoretical advantages compared to plain Autoencoders as discussed in Section 3.2.2. Most importantly, they offer a probabilistic interpretation of the latent space. The encoder learns to produce $p(z|X)$ as a multivariate Gaussian with mean and variance parametrization. This distribution is regularized by the KL-divergence loss to approach zero mean and unit variance. Practically, the regularization ensures that the latent codes are densely distributed. The sampling procedure during training ensures that the latent space around the mean values represents similar data points, i.e. similar looking object views. A pure maximum likelihood method like the plain Autoencoder would theoretically allow to store similar orientations at very different latent codes.

When training with synthetic data, we randomly sample from an infinite number of views. In VAEs, the latent representations are imposed with Gaussian noise. However, the infinite sampling and the random augmentations can also be interpreted as perturbations of the latent representation. The variable data source seems to produce a continuously meaningful latent space by itself.

The plain Autoencoder does not regularize the expansion of latent codes. However, in the case of orientation learning the property of a dense latent space seems to be inherent through the closed circular nature of the rotational space, as shown in Figure 4.7.

Experiments have shown that the variance produced by the encoder of the VAE is very small ($\Sigma \ll 1$) for any inputs. It behaves like a plain Autoencoder. Increasing the KL-divergence term through a factor, as proposed in the beta VAE [16], increases the variance but hurts the performance. Decreasing the factor does not change results. These findings motivate the use of the plain AE as it is simpler to implement. Under the same training conditions the results of VAE and AE are very similar (Table 4.10), even though the loss functions are quite different. L2 loss for the AE and Bernoulli + KL loss for the VAE.

Table 4.10: AE vs. VAE, synthetic data

	Primesense	Kinect
VAE	0.914 (± 0.003)	0.931 (± 0.001)
AE	0.910 (± 0.004)	0.930 (± 0.001)

This shows that the plain Autoencoder, in our case, is sufficient for training with synthetic data. Nonetheless, there are scenarios where the VAE has superior performance. Considering the training with few examples of synthetic or real data, the VAE outperforms the plain AE. Therefore, if a small amount of pose annotated real sensor data is available, the training should be preferably executed on a VAE.

4.7 Joint object detection and 3D orientation evaluation

The augmented Autoencoders have proven to reliably predict 3D orientations of rigid objects from ground truth RGB crops. Now, it is examined how they perform in combination with scene crops from the SSD object detector. The absolute depth of the object is assumed to be given at this point. An Iterative Closest Point (ICP) algorithm and/or additional depth data could be used to estimate the depth to determine the full 6D pose transformation.

If t_z is known, t_x and t_y components can be simply obtained from the 2D detection and the camera calibration parameters using the pinhole camera model. Therefore, the pixel offset from the principal point p to the predicted bounding box center bb_{cent} is determined. Using

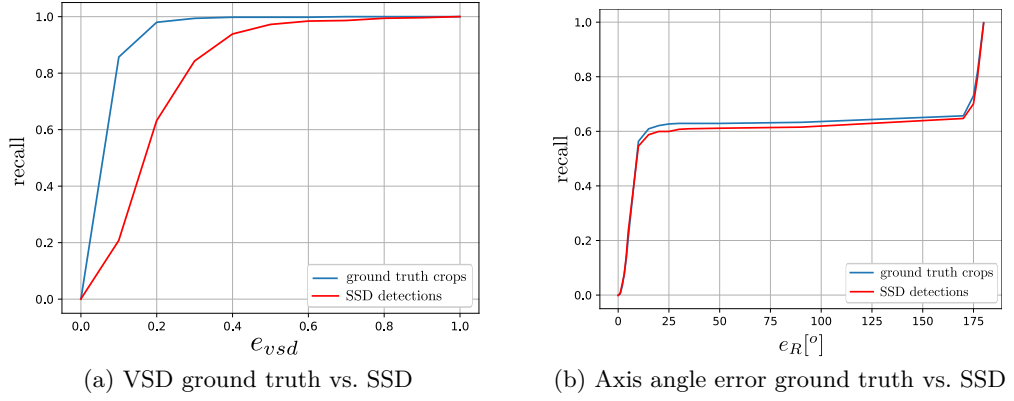


Fig. 4.9: Influence of 2D detection on pose error metrics (Kinect)

the focal length f of the camera allows to determine the full translation vector from camera to object.

$$\begin{pmatrix} t_x \\ t_y \end{pmatrix} = \frac{t_z}{f} \begin{pmatrix} bb_{cent,x} - p_x \\ bb_{cent,y} - p_y \end{pmatrix} \quad (4.8)$$

During training the Autoencoders have to revert random translations of the object from the image center. This augmentation is supposed to make the orientation predictions robust against uncertainties in the 2D detection.

Figure 4.9 and Table 4.11 describe the joint performance again on all 504 views of object 5 in scene 2.

Table 4.11: AUC values of VAE with and w/o 2D detection

	Kinect			Primesense		
	AUC _{re}	AUC _{re,rect}	AUC _{vsd}	AUC _{re}	AUC _{re,rect}	AUC _{vsd}
ground truth crops	0.575	0.916	0.931	0.574	0.907	0.914
SSD crops	0.560	0.919	0.802	0.525	0.875	0.747

It can be observed that the VSD error rises significantly due to imperfect detections while the rotation errors are hardly affected. Therefore, it can be assumed that the lower AUC_{vsd} almost exclusively stems from the translational errors of the detector. Eventually, it proves the stability of the orientation estimation with respect to translational offsets.

For a qualitative insight, Figure 4.10 depicts 18 views of scene 11 where object 5 is detected and the 3D orientation is estimated. The results are quite stable under moderate occlusions, despite the texture-less object surface. In the last row, two of the orientation predictions are incorrect presumably due to occlusions. In the appendix, more objects and scene sequences can be found (Figure A.3, A.4, A.5).

Unfortunately, a quantitative comparison to other pose estimation approaches is non-trivial. The large number of evaluation metrics are mostly designed for full 6D pose estimation which is not considered here. In principle, any template-based depth estimation method could be plugged in, e.g. [Iterative Closest Point \(ICP\)](#) or CAD-based edge matching [57], reducing the problem to a line search. However, it is not the focus of this thesis and a naive depth estimation would not yield competitive results. T-LESS is also a new dataset with very few published results [23, 47]. Therefore, a throughout comparison is left to future work.

4.8 Variants

4.8.1 Conditional Variational Autoencoder

The Conditional Variational Autoencoder (CVAE) is an attempt to train only one model to learn the orientation of multiple objects. Table 4.12 depicts the performance of the CVAE trained in parallel on four T-LESS objects (5,8,9,10). The tests are executed on scene 11 by creating four training embeddings, one for each object. Below the performance of single VAEs is depicted.

Table 4.12: AUC_{vsd} on scene 11, objects

	object 5	object 8	object 9	object 10
CVAE	0.862	0.823	0.878	0.799
VAE	0.886	0.861	0.891	0.802

It can be seen that the AUC_{vsd} values slightly decline. The training time rises with the number of objects. Here, the accuracies saturate at iteration ~ 45000 . Since this diminishes the benefit of shorter training time, multiple single object models might be preferable for highest accuracy. On the other hand, it shows the ability of the approach to learn multiple representations in parallel.

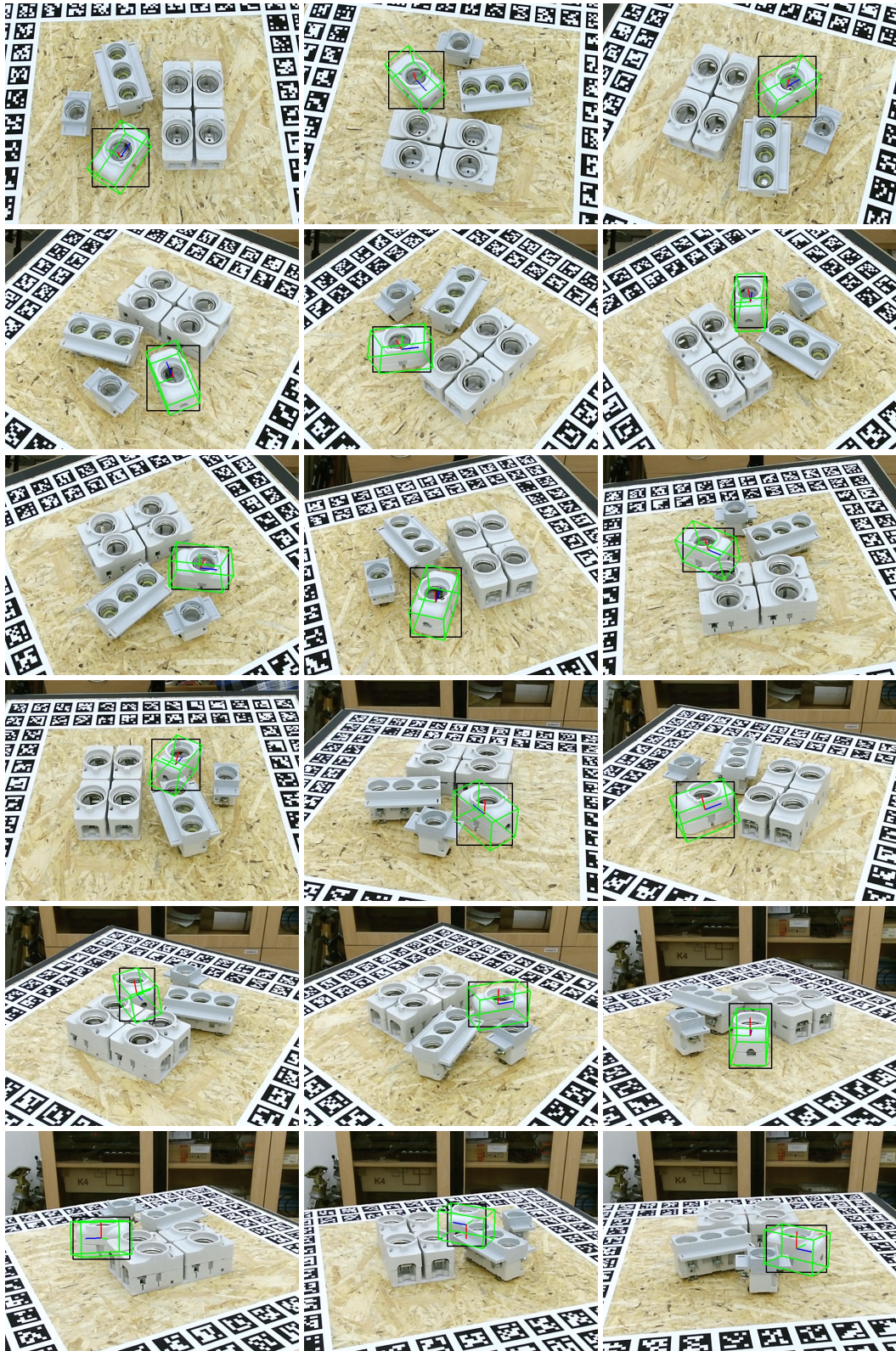


Fig. 4.10: 2D detection (black) and 3D orientation estimation (green) of object 5; tested on every 20th view of T-LESS scene 11

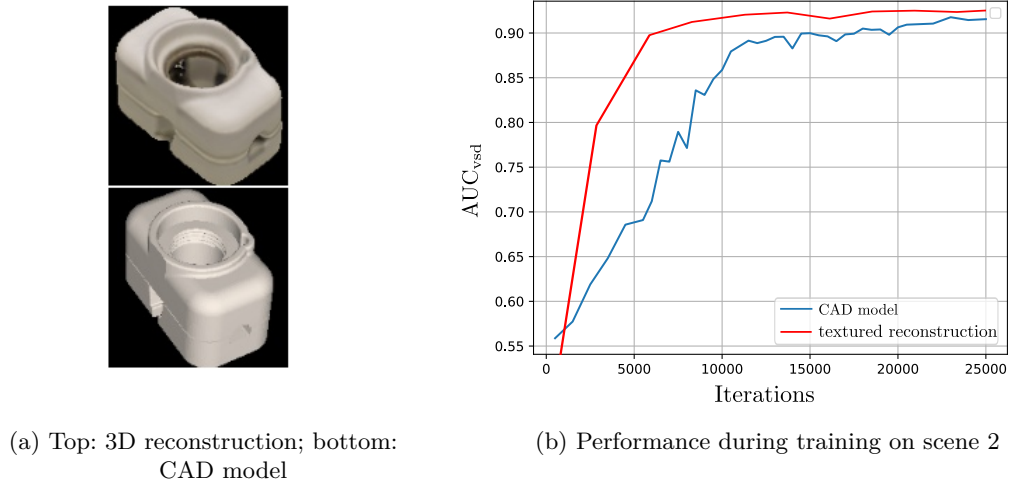


Fig. 4.11: Training on CAD model vs. textured 3D reconstruction

4.8.2 Training on CAD models

The [AE](#) can also be trained on synthetic views of raw CAD models, see bottom of Figure [4.11a](#). Although, the training takes longer to converge, it almost reaches the same performance as the training with the textured 3D model reconstruction. The development of the AUC_{vsd} during training is depicted in Figure [4.11b](#). The model, training and augmentation configurations (Table [4.7](#)) are equivalent.

This result is remarkable since important color information is neglected in the CAD model. The presented augmentations still manage to avoid over-fitting on specific color information and shape features can still be recognized in the real sensor data. For texture-less objects like in T-LESS, the generalization works surprisingly well.

Naturally, training on CAD models is expected to work less well for highly textured objects. Depending on the type of object, a simple colorization of parts could suffice to train smooth color regions.

5 Future work

The potential of the [Autoencoder \(AE\)](#) in orientation estimation has been shown. Certainly, a future goal is to extend the approach to full 6D pose estimation.

First experiments to train the [AE](#) with synthetic RGB-D data have not improved the results over plain RGB. In future, augmentations towards the generalization of synthetic depth data to real sensor depth data should be systematically investigated.

Furthermore, the object views could be generated at different radii. Then, the encoder can be trained to either incorporate the object depth into the embedding or to directly regress the object depth using an additional encoder output and loss. The goals of orientation and depth estimation should work well together and the performance could potentially benefit from multi-task learning.

The capabilities of the approach as a general feature extractor for object orientation estimation should be further examined. Preliminary experiments showed that the training and testing embeddings can be produced by other objects than the one used for training the weights of the [AE](#). This might be an interesting approach towards zero-shot learning [\[63\]](#).

Interpolating the orientations of the [kNN](#) in the training embedding potentially improves the performance. However, it is not guaranteed to work in the case of certain symmetries in the object. Therefore, a clustering of orientations (rotation matrices) has to be performed in advance to ensure that only close orientations are interpolated.

Adversarial Autoencoders [\[40\]](#) or similar approaches could increase the clarity of the reconstructions such that smaller details in the latent codes are included.

The object detector [SSD](#) has not performed very well on synthetic data, yet. This is presumably because of over-fitting to synthetic features. Even though there is not a direct way to teach [SSD](#) invariances as in the [AE](#), stronger training augmentations could still increase the generalization.

Finally, in the long-term a single feature extractor [CNN](#) should be trained to extract features for the full 6D object pose estimation. As in [SSD](#), different head networks (e.g. decoders) could interpret these features to determine different pose dimensions.

6 Conclusion

In this thesis, a novel pipeline for joint 2D object detection and 3D orientation estimation on RGB data has been presented.

For detection, the Single Shot Multibox Detector [45] has been fine-tuned and evaluated on the T-LESS dataset.

Traditional approaches for object orientation estimation are often restricted to certain object characteristics or environment conditions. Learning-based approaches can extract more complex and robust features, but usually rely on pose annotated sensor data which is difficult to obtain in robotic applications.

Based on the challenges identified in this thesis, an [Autoencoder \(AE\)](#) CNN architecture for object orientation estimation has been proposed. Utilizing a new training procedure, the [AE](#) learns low-dimensional representations for all possible object orientations in RGB images.

The approach has several advantages:

- The representations can be explicitly trained to become invariant against irrelevant illumination, background clutter, translations and occlusions. This is achieved by applying artificial augmentations on the [AE](#) input during training while the goal is to reconstruct clean output views.
- The augmentation technique also allows to train the [AE](#) using synthetic views of textured 3D models or even raw CAD models. Thus, no manual 3D orientation annotations of sensor data are required.
- In contrast to state-of-the-art approaches, the proposed pipeline runs in only 24ms on a modern GPU. It is therefore well suited for real-time applications.

The effectiveness of the individual augmentations has been evaluated on the T-LESS dataset using different error metrics. It has been shown that the training on synthetic views generalizes well on RGB recordings of three different sensors. Furthermore, the accuracy of the 3D orientation estimations hardly decreases with inaccurate object detections.

Potential applications are in dynamic environments where objects frequently appear and leave the field of view. For example, an object tracking algorithms could be reinitialized by our method when loosing track. Incorporating the depth estimation would also allow fast robotic manipulation and assembly.

To conclude, the main contribution of this thesis is a new, simple and efficient approach for 3D orientation estimation trained purely on synthetic data.

Bibliography

- [1] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.
- [2] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother. Learning 6d object pose estimation using 3d object coordinates. In *European Conference on Computer Vision*, pages 536–551. Springer, 2014.
- [3] D. A. Butler, S. Izadi, O. Hilliges, D. Molyneaux, S. Hodges, and D. Kim. Shake’n’sense: reducing interference for overlapping structured light depth cameras. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1933–1936. ACM, 2012.
- [4] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. *arXiv:1602.02481*, 2016.
- [5] A. Crivellaro, M. Rad, Y. Verdie, K. Moo Yi, P. Fua, and V. Lepetit. A novel representation of parts for accurate 3d object detection and tracking in monocular images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4391–4399, 2015.
- [6] P. David, D. DeMenthon, R. Duraiswami, and H. Samet. Simultaneous pose and correspondence determination using line features. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2003.
- [7] A. Doumanoglou, V. Balntas, R. Kouskouridas, and T.-K. Kim. Siamese regression networks with efficient mid-level feature extraction for 3d object pose estimation. *arXiv preprint arXiv:1607.02257*, 2016.
- [8] S. Ekvall and D. Kragic. Learning and evaluation of the approach vector for automatic grasp generation and planning. In *Robotics and Automation, 2007 IEEE International Conference on*, pages 4715–4720. IEEE, 2007.
- [9] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. <http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html>.

- [10] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2): 303–338, 2010.
- [11] N. I. Fisher. *Statistical analysis of circular data*. Cambridge University Press, 1995.
- [12] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.
- [13] I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.
- [14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 770–778, 2016.
- [15] K. He, G. Gkioxari, P. Dollár, and R. Girshick. Mask r-cnn. *arXiv preprint arXiv:1703.06870*, 2017.
- [16] I. Higgins, L. Matthey, A. Pal, C. Burgess, X. Glorot, M. Botvinick, S. Mohamed, and A. Lerchner. beta-vae: Learning basic visual concepts with a constrained variational framework. 2016.
- [17] S. Hinterstoisser, S. Benhimane, V. Lepetit, P. Fua, and N. Navab. Simultaneous recognition and homography extraction of local patches with a simple linear classifier.
- [18] S. Hinterstoisser, S. Holzer, C. Cagniart, S. Ilic, K. Konolige, N. Navab, and V. Lepetit. Multimodal templates for real-time detection of texture-less objects in heavily cluttered scenes. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 858–865. IEEE, 2011.
- [19] S. Hinterstoisser, C. Cagniart, S. Ilic, P. Sturm, N. Navab, P. Fua, and V. Lepetit. Gradient response maps for real-time detection of textureless objects. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 34(5):876–888, 2012.
- [20] S. Hinterstoisser, V. Lepetit, S. Ilic, S. Holzer, G. Bradski, K. Konolige, and N. Navab. Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes. In *Asian conference on computer vision*, pages 548–562. Springer, 2012.
- [21] T. Hodan. SIXD Challenge 2017, 2017. URL http://cmp.felk.cvut.cz/sixd/challenge_2017/.
- [22] T. Hodan, J. Matas, and S. Obdržálek. On evaluation of 6d object pose estimation.

-
- [23] T. Hodaň, P. Haluza, Š. Obdržálek, J. Matas, M. Lourakis, and X. Zabulis. T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017.
- [24] J. Huang, V. Rathod, C. Sun, M. Zhu, A. Korattikara, A. Fathi, I. Fischer, Z. Wojna, Y. Song, S. Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. *arXiv preprint arXiv:1611.10012*, 2016.
- [25] W. Kehl, F. Milletari, F. Tombari, S. Ilic, and N. Navab. Deep learning of local rgb-d patches for 3d object detection and 6d pose estimation. In *European Conference on Computer Vision*, pages 205–220. Springer, 2016.
- [26] A. Kendall, M. Grimes, and R. Cipolla. Posenet: A convolutional network for real-time 6-dof camera relocalization. In *Proceedings of the IEEE international conference on computer vision*, pages 2938–2946, 2015.
- [27] D. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.
- [28] D. P. Kingma and M. Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013.
- [29] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.
- [30] A. Krull, F. Michel, E. Brachmann, S. Gumhold, S. Ihrke, and C. Rother. 6-dof model based tracking via object coordinate regression. In *Asian Conference on Computer Vision*, pages 384–399. Springer, 2014.
- [31] A. Krull, E. Brachmann, F. Michel, M. Ying Yang, S. Gumhold, and C. Rother. Learning analysis-by-synthesis for 6d pose estimation in rgb-d images. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 954–962, 2015.
- [32] T. D. Kulkarni, W. F. Whitney, P. Kohli, and J. Tenenbaum. Deep convolutional inverse graphics network. In *Advances in Neural Information Processing Systems*, pages 2539–2547, 2015.
- [33] A. B. L. Larsen, S. K. Sønderby, H. Larochelle, and O. Winther. Autoencoding beyond pixels using a learned similarity metric. *arXiv preprint arXiv:1512.09300*, 2015.
- [34] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010. URL <http://yann.lecun.com/exdb/mnist/>.
- [35] V. Lepetit, F. Moreno-Noguer, and P. Fua. Epnp: An accurate $\mathcal{O}(n)$ solution to the pnp problem. *International Journal Computer Vision*, 81(2), 2009.

- [36] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *European Conference on Computer Vision*, pages 21–37. Springer, 2016.
- [37] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [38] D. G. Lowe. Three-dimensional object recognition from single two-dimensional images. *Artificial intelligence*, 31(3):355–395, 1987.
- [39] J. B. Madsen and R. Stenhold. How wrong can you be: Perception of static orientation errors in mixed reality. In *3D User Interfaces (3DUI), 2014 IEEE Symposium on*, pages 83–90. IEEE, 2014.
- [40] A. Makhzani, J. Shlens, N. Jaitly, I. Goodfellow, and B. Frey. Adversarial autoencoders. *arXiv preprint arXiv:1511.05644*, 2015.
- [41] F. Michel, A. Krull, E. Brachmann, M. Y. Yang, S. Gumhold, and C. Rother. Pose estimation of kinematic chain instances via object coordinate regression. In *BMVC*, pages 181–1, 2015.
- [42] R. A. Newcombe, S. Izadi, O. Hilliges, D. Molyneaux, D. Kim, A. J. Davison, P. Kohi, J. Shotton, S. Hodges, and A. Fitzgibbon. Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.
- [43] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 343–352, 2015.
- [44] E. Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.
- [45] B. Paul. SSD Tensorflow implementation, 2017. URL <https://github.com/balancap/SSD-Tensorflow>.
- [46] B. T. Phong. Illumination for computer generated pictures. *Communications of the ACM*, 18(6):311–317, 1975.
- [47] M. Rad and V. Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. *arXiv preprint arXiv:1703.10896*, 2017.
- [48] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

-
- [49] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.
 - [50] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.
 - [51] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb. Learning from simulated and unsupervised images through adversarial training. *arXiv preprint arXiv:1612.07828*, 2016.
 - [52] H. A. Siddharth Mahendran and R. Vidal. 3d pose regression using convolutional neural networks.
 - [53] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
 - [54] K. Sohn, H. Lee, and X. Yan. Learning structured output representation using deep conditional generative models. In *Advances in Neural Information Processing Systems*, pages 3483–3491, 2015.
 - [55] J. T. Springenberg, A. Dosovitskiy, T. Brox, and M. Riedmiller. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.
 - [56] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.
 - [57] M. Ulrich, C. Wiedemann, and C. Steger. Cad-based recognition of 3d objects in monocular images. In *ICRA*, volume 9, pages 1191–1198, 2009.
 - [58] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, and P.-A. Manzagol. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *Journal of Machine Learning Research*, 11(Dec):3371–3408, 2010.
 - [59] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features. In *Proceedings of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001*, volume 1, pages I–511–I–518 vol.1, 2001. doi: 10.1109/CVPR.2001.990517.
 - [60] T. Weise, T. Wismer, B. Leibe, and L. Van Gool. In-hand scanning with online loop closure. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1630–1637. IEEE, 2009.

- [61] C. Wöhler. Three-dimensional pose estimation and segmentation methods. In *3D Computer Vision*, pages 89–137. Springer, 2013.
- [62] P. Wohlhart and V. Lepetit. Learning descriptors for object recognition and 3d pose estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3109–3118, 2015.
- [63] Y. Xian, B. Schiele, and Z. Akata. Zero-shot learning - the good, the bad and the ugly. *CoRR*, abs/1703.04394, 2017. URL <http://arxiv.org/abs/1703.04394>.

Figures

1.1	DLR Justin	1
1.2	Coordinate transformation	1
1.3	Proposed object pose estimation pipeline	3
2.1	SingleShot Multibox Detector architecture; <i>Note.</i> adapted from [36]	9
2.2	All 30 T-LESS objects in ascending order	10
3.1	Overview of different object symmetries: axis+plane, 3-planes, point	15
3.2	Symmetric and almost symmetric views	15
3.3	Autoencoders for clustering and denoising	18
3.4	Variational Autoencoder architecture and re-parametrization trick	21
3.5	Variational Autoencoder CNN architecture with input/output at test time; (simplified sampling, see Fig. 3.4!)	23
3.6	T-LESS object 5 recording; left: Primesense original, middle: 3D reconstruction from Primesense, right: Canon camera scene crop	24
3.7	Latent space of synthetic and real views after successful training	25
3.8	Impact of scale augmentation on embedding; right: 36 binary masks at $scale = 0.7$; top left: VAE $z \in \mathcal{R}^2$ trained at $scale = 1.0$; bottom left: VAE $z \in \mathcal{R}^2$ trained at $scale \sim \mathcal{U}(0.7, 1.0)$	27
3.9	Left: Clean virtual view of textured 3D model of object 5 (desired output); middle: augmented input; right: decoder reconstruction after 35000 iterations	28
3.10	VAE orientation estimation; top: offline sampling and generation of training embedding; bottom: online inference of scene crops; right: nearest neighbor calculation	30
4.1	Single Shot Multibox Detector; a) example scene; b) precision-recall statistics	36
4.2	Axis-angle error histogram: object 5, Canon scene 2, 4° intervals	38
4.3	Interpolated cumulative error-recall curves for Top $k \in \{1, 3, 5, 10, 50\}$ NNs	38
4.4	VAE trained with and without occlusion augmentation	41
4.5	Reconstruction of Kinect RGB crop of scene 11 and synthetic Nearest Neighbor views	42
4.6	Dependency of performance on latent space size (Kinect) standard deviation in red	43
4.7	3 Principle Components of test latent space of scene 2, object 5	44

4.8	Decoder reconstruction of a test latent vector scaled by factor $s \in [0, 2.5]$	45
4.9	Influence of 2D detection on pose error metrics (Kinect)	47
4.10	2D detection (black) and 3D orientation estimation (green) of object 5; tested on every 20th view of T-LESS scene 11	49
4.11	Training on CAD model vs. textured 3D reconstruction	50
A.1	SSD average-precision for objects {2,3,7,9} in scene 12 trained on synthetic data	65
A.2	Rotation error histogram of a CNN classifier trained on the 1296 Canon views (as classes) of object 5, scene 11, without considering in-plane rotations. Since the object lies upright on the table, the classifier only needs to predict azimuth and elevation angles. In this setting it works decently ($avg(e_R) = 12.31^\circ$) and even detects the symmetry-breaking notch. However, it is not scalable to a fine-grained full 3D rotational space.	65
A.3	Incomplete IKEA mug 3D orientation estimation from webcam stream (left), nearest training neighbors (right)	66
A.4	2D detection (black) and 3D orientation estimation (green) of object 30; tested on scene 1	66
A.5	2D detection (black) and 3D orientation estimation (green) of object 2; tested on scene 12; note that there are two instances	67
A.6	2D detection (black) and 3D orientation estimation (green) of object 9; tested on scene 5 with ground truth bounding boxes	68

Tables

3.1	Inference time of pipeline blocks	32
4.1	SSD characteristics	35
4.2	Average Precision SSD	36
4.3	AUCs for VAE training with Canon images	39
4.4	Input color augmentation strengths of experiments in 4.6	40
4.5	Input shape augmentation strengths of experiments in 4.6	40
4.6	Ablation study on color augmentations	40
4.7	Stronger augmentation	42
4.8	AUCs for VAE training with synthetic images	42
4.9	Best VAE model with $z \in R^{128}$	43
4.10	AE vs. VAE, synthetic data	46
4.11	AUC values of VAE with and w/o 2D detection	47
4.12	AUC _{vsd} on scene 11, objects	48

Appendix

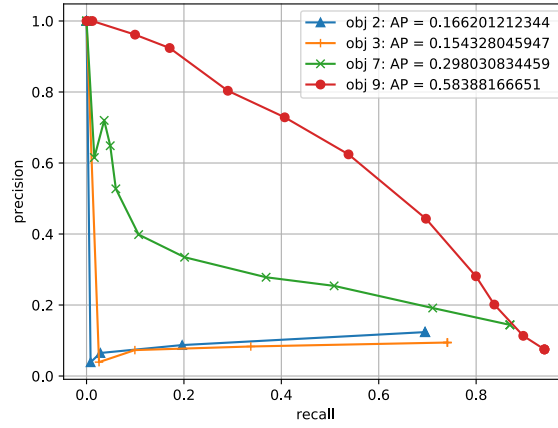


Fig. A.1: SSD average-precision for objects $\{2,3,7,9\}$ in scene 12 trained on synthetic data

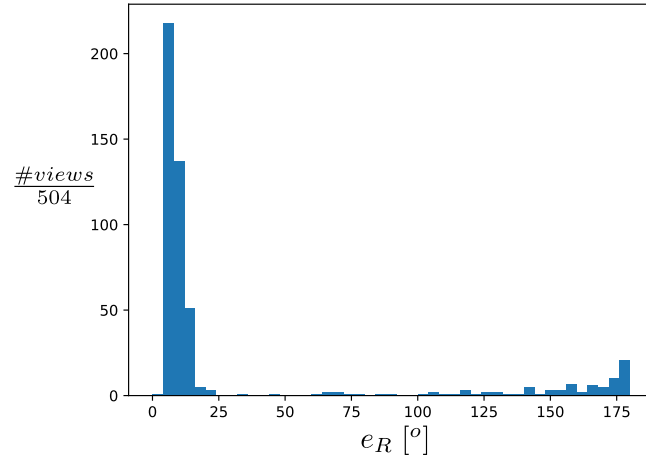


Fig. A.2: Rotation error histogram of a CNN classifier trained on the 1296 Canon views (as classes) of object 5, scene 11, **without** considering in-plane rotations. Since the object lies upright on the table, the classifier only needs to predict azimuth and elevation angles. In this setting it works decently ($avg(e_R) = 12.31^\circ$) and even detects the symmetry-breaking notch. However, it is not scalable to a fine-grained full 3D rotational space.



Fig. A.3: Incomplete IKEA mug 3D orientation estimation from webcam stream (left), nearest training neighbors (right)

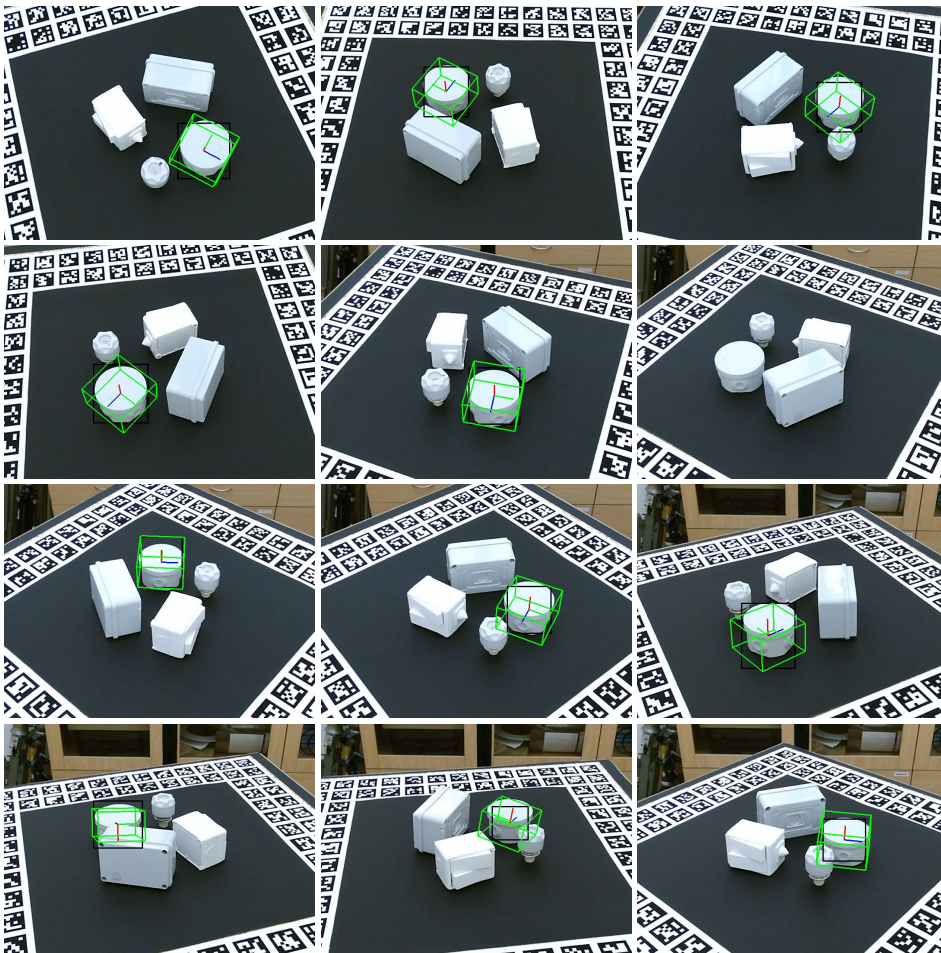


Fig. A.4: 2D detection (black) and 3D orientation estimation (green) of object 30; tested on scene 1



Fig. A.5: 2D detection (black) and 3D orientation estimation (green) of object 2; tested on scene 12; note that there are two instances

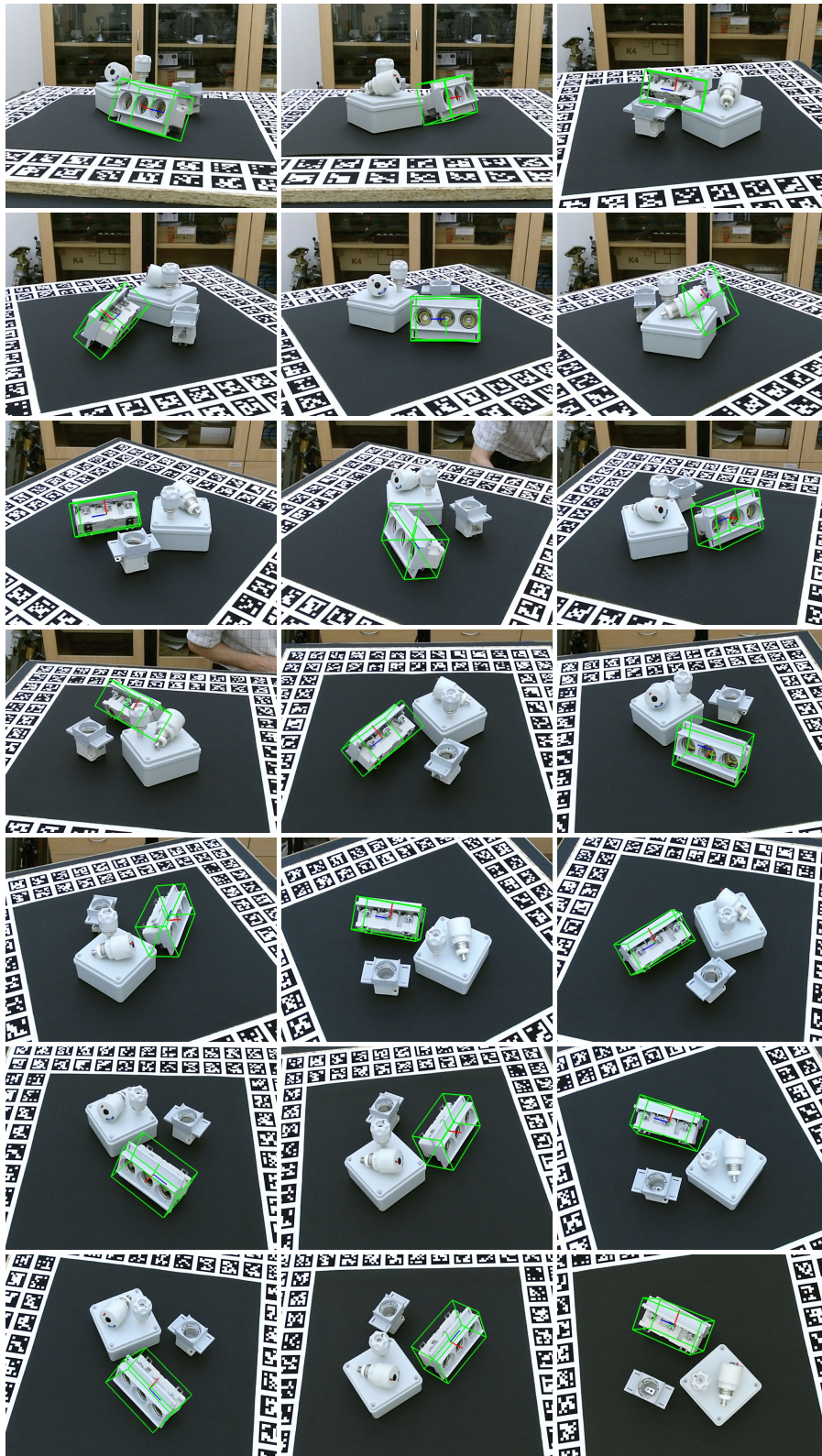


Fig. A.6: 2D detection (black) and 3D orientation estimation (green) of object 9; tested on scene 5 with ground truth bounding boxes