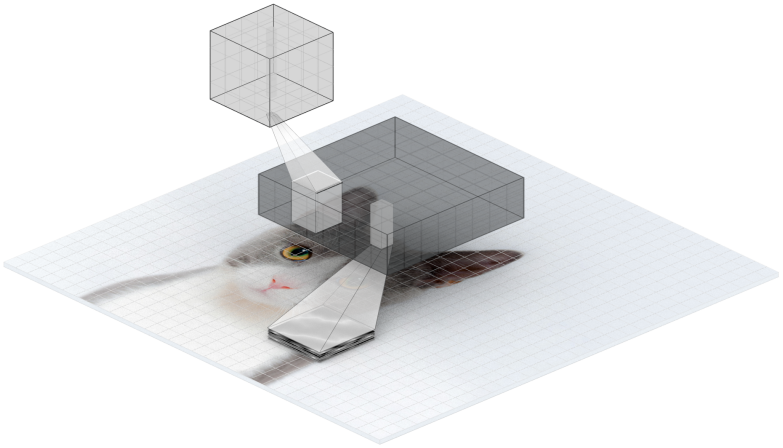
Imperial College London

Department of Electrical and Electronic Engineering

Final Year Project Report 2018



Project Title:	**RGBD Feature Learning for 6D Object Pose Estimation**
Student:	**Tomas Pulmann**
CID:	**00976353**
Course:	**EE4**
Project Supervisor:	**Dr Tae-Kyun Kim**
Second Marker:	**Dr Krystian Mikolajczyk**

Abstract

Estimating the 6D pose of known objects is a task of great interest due to the many potential applications in fields such as robotics and augmented reality. While deep neural networks have become the state-of-the-art in related tasks such as 2D object detection, 6D pose estimation has shown to be challenging because of the increased complexity associated with inference in 3D space and pose ambiguity due to factors such as symmetry and occlusion. It has been established that utilizing the depth channel of an image, if available, can increase pose estimation accuracy in challenging scenarios. However, the majority of existing approaches have relied on using depth to refine pose estimates using analytic optimization-based techniques, rather than integrating depth and color channels within a single network. This work's primary contribution is adapting the deep learning state-of-the-art 2D detection system known as Faster R-CNN for the task of 6D object pose estimation using RGBD data, resulting in a novel, end-to-end deep learning architecture. Secondarily, we demonstrate successful transfer of domain knowledge by adapting an RGB feature extractor for the depth channel. The resulting system achieves higher performance compared to an RGB-only baseline on several standard pose accuracy metrics, and in some cases, approaches the state-of-the-art despite being trained for a fraction of the time.

Acknowledgements

First and foremost, I would like to thank my supervisor, Dr. Tae-Kyun Kim, for providing his invaluable guidance throughout the project. I would also like to extend my gratitude to Sock Ju-il of the Imperial Computer Vision & Learning Lab for his advice and recommendations that helped define my work along the way.

Contents

List of Tables

List of Figures

Chapter 1

Introduction and Projection Specification

Computer Vision has become a highly active area of research in recent years. In general terms, the field is concerned with extracting useful abstract information from visual inputs such as images and video. The problem is not a straightforward one - in digital computers, images are represented as large arrays of intensity values, and the challenge is to extract high-level information such as object classes and positions, scene segments, etc. given this raw visual data. The field dates back to 1966, when MIT researchers expected to solve these tasks over the course of a summer project [6]. It soon became apparent that computer vision is much more challenging than previously assumed, and it took decades of research until systems with performance comparable to humans were created. In 2012, deep neural networks demonstrated a breakthrough in object classification accuracy [7], and the fields of deep learning and computer vision have been getting increasingly interconnected since then.

1.0.1 Neural Networks and Deep Learning

Artificial Neural Networks (ANNs) are machine learning models consisting of interconnected units which make individual decisions by weighing several inputs, analogically to biological

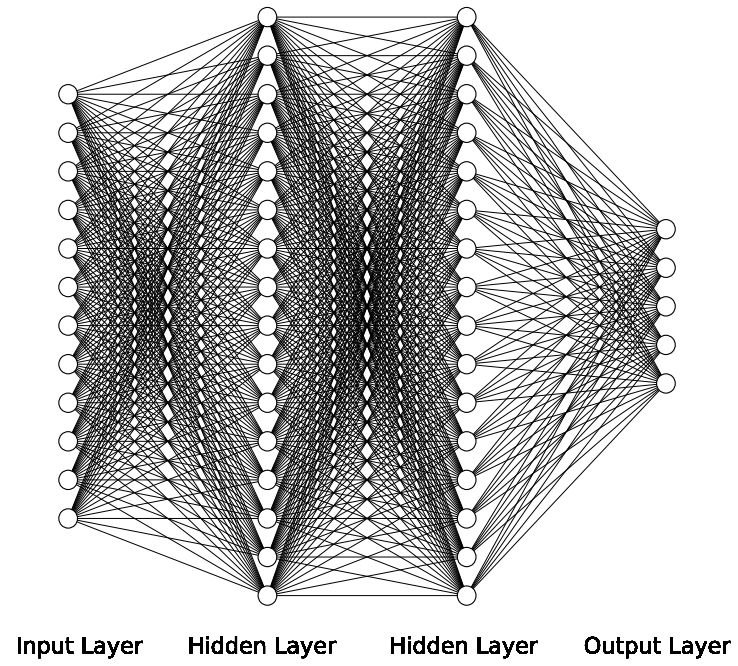


Figure 1.1: A small fully-connected artificial neural network. It is apparent that a deep network of this type with a high-dimensional input space would need an enormous amount of parameters to represent all of the neural weights. To overcome this issue, practical deep neural networks use many convolutional layers with much fewer parameters, and only a few of the topmost layers tend to be fully-connected.

neurons. Unlike the Perceptron [8], which maps directly from input space to a binary output, ANNs contain hidden layers which allow them to create useful intermediate representations. The activation of each artificial neuron in a given layer is a weighted sum of activations from the previous layer that is subsequently transformed by a non-linear function. In practice, a very simple non-linearity $f(x) = max(0, x)$ has been shown to perform very well [7] compared to more complicated non-linear functions. Neural networks are usually trained by gradient descent methods, which requires defining a loss function that is subsequently minimized by adjusting the network's weights iteratively.

Deep neural networks are a subset of ANNs that have many layers stacked on top of each

other. There is no clear boundary in terms of the number of layers that separates deep neural networks from the rest, but modern networks can be very deep - Resnet [9] achieved state-of-the-art performance on image classification by utilizing 152 layers in total. Deep neural networks learn hierarchical representations, building up complicated concepts on top of simpler ones [10]. One class of a particularly successful model in the field of Computer Vision is the Convolutional Neural Network (CNN) [11].

In fully connected neural networks, each output in layer n takes as input all of the outputs in layer $n-1$, and each connection has an associated weight. In CNNs, a certain output is only connected to a small region of the previous layer - inspired loosely by the human visual system, the limited spatial extent of the artificial neuron is often referred to as its receptive field. The set of parameters that map from the input region to an activation is called a filter, and one such filter contains much fewer parameters than a fully-connected layer. The output activation map is obtained by sliding a filter across the input layer. Mathematically, this operation corresponds to the *convolution* of the input layer with the filter's parameters. Typically, many such filters are convolved with a single layer, each producing its own activation map. In practice, convolutional layers are often followed by pooling layers, which reduce feature map dimensions by taking the maximum or average over small regions, e.g. 2×2.

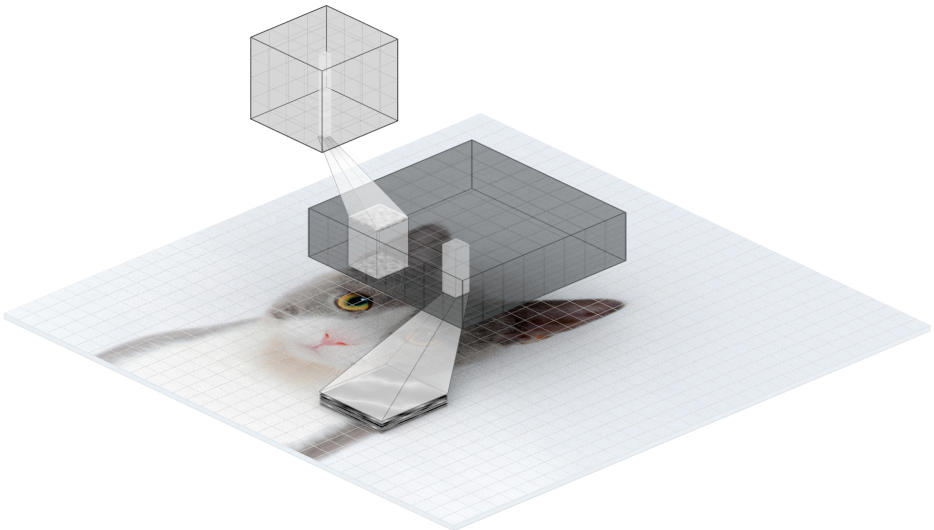


Figure 1.2: A Convolutional Neural Network with 2 layers stacked on top of each other. The layers are depicted as having depth to emphasize that many different filters are convolved with the previous layer.

1.0.2 6D Object Pose Estimation

This work is concerned with applying deep learning to the problem of estimating the 6D pose of objects. The pose of a rigid object is denoted as 6D because it has 6 degrees of freedom - 3 in translation and 3 in rotation. The goal is to infer these quantities from RGBD images, where the individual channels correspond to red, green, blue and depth. With the rise of affordable depth sensors, RGBD cameras such as Microsoft Kinect have enabled collection of rich datasets of such images. Given such images as input, the task of a 6D Pose Estimation system is to infer 3D position and orientation of objects in an image. Formally, the 4×4 matrix of interest is:

$$\mathbf{P} = [\mathbf{R}, \mathbf{t}, 0, 1] \tag{1.1}$$

As defined in [2]. This matrix transforms an arbitrary point $\mathbf{p} = (x, y, z, 1)^T$ in homogeneous coordinates from the model's coordinate system to a particular position within the camera's coordinate system. The matrix $\mathbf{R}$ is a 3×3 rotation matrix that can be parametrized by an axis l and an angle of rotation θ, while $\mathbf{t}$ is a 3×1 translation vector relative to the observing camera's origin. There are two distinct variants of the 6D pose estimation problem:

1. *6D Pose Localization*: Prior information about the number of objects and their respective classes in the image is available.

2. *6D Pose Detection*: No prior information about the number of objects or class occurrences is available.

The problem of interest in this project is a special case of the latter task, where 2 or 3 instances of a given object are present in an image. From a practical point of view, 6D object pose estimation is of significant interest due to the vast number of applications in fields such as robotics (picking objects requires knowing their pose) and augmented reality (object poses are needed for seamless integration of virtual and real elements).

1.0.3 Project Specification

Other than the general final year project deliverables, this work has 2 other major deliverables:

1. A trained and evaluated 6D Object Pose Estimation system based on an existing deep learning system called Faster R-CNN. The system must take as input RGB images and output the predicted 6D pose of the detected objects of interest.

2. An extension of the above network that uses depth data in addition to the color channels. The system must take as input RGBD images and output the predicted 6D pose of the detected objects of interest.

The purpose of the RGB-only network is to serve as a baseline for comparison with the depth-capable one. As part of project planning defined in the interim report, projected completion dates were chosen for the major project milestones. These projections as well as actual completion dates of the milestones are outlined in table 1.1 below.

Milestone	Projected Completion Date	Actual Completion Date
Faster R-CNN Implementation	January 2018	February 2018
6D Pose Estimation Network (RGB only)	February 2018	March 2018
6D Pose Estimation Network (RGBD)	April 2018	May 2018

Table 1.1: 6D pose estimation network implementation milestones.

Although the completion dates lag the projected dates by a month on all 3 milestones, this is not a concern as the project plan was defined conservatively in order to provide a safety margin. Hence, sufficient time was reserved for evaluation even in case of delays. Following the successful implementations of both pose estimation network variants, the primary objective of this work is to investigate whether the addition of the depth channel results in a performance increase in the fully learned, end-to-end approach to the problem. This report first introduces the relevant background research, implementation details, experimental results and their analysis with respect to the objective.

Chapter 2

Background Research

2.1 Related Work

A wide variety of approaches have been researched in the context of object pose estimation. In general terms, most of the traditional successful methods rely on extracting image features, either densely (pixel-wise) or sparsely (patch-wise), and matching them to a known database [12]. In the template-based approach, a particular object pose is simply represented by a template image of the object in that specific pose. If a location inside a test image matches the template, it counts as a successful object detection in the given pose. One of the most successful methods within this paradigm utilized so-called LINE-MOD [3] features based on edge color gradients and surface normals estimated from depth maps to obtain object pose estimates robust to varying lighting conditions and background clutter. During testing, extracted features are matched against the templates to get a coarse estimate of the pose, which is subsequently refined using the iterative closes point (ICP) algorithm [13]. This technique was also successful on textureless objects due to the incorporation of depth data, since the color channel fails to provide discriminative features for this type of objects in regions other than the edges.

Another contribution of this work was the utilization of synthetic data for training. Sufficient coverage of the pose space is a great challenge for real image data, and the collection and annotation of an exhaustive enough dataset is highly time-consuming. The alternative approach

employing synthetic data involves rendering the object from many views to obtain rich pose-space coverage. This method provides the benefit of precise control and easy automatic ground-truth annotation: the authors of [3] sampled the views from vertices of the upper hemisphere of an icosahedron of varying scales.

The disadvantage of template methods stems from the fact that a very high amount of templates must be generated in order to achieve sufficient pose space coverage and scale invariance. This approach is considered to be holistic, meaning that its accuracy is directly tied to the correspondence of entire objects. As a consequence, the accuracy of this paradigm suffers in the presence of occlusion and substantial noise [14], exhibiting an approximately linear relationship between object surface visibility and pose estimation accuracy [3]. On the upside, the methods exhibit quick runtime, making them sufficient for real-time applications.

To address the weaknesses of the methods above, several researchers have turned to approaches based on local correspondences. In general, these approaches extract local image patches which are treated independently to construct object pose hypotheses. While these methods do not take global context into account, they aggregate many local correspondences to achieve robustness to occlusion and noise. Some works within this paradigm have taken the route of sparse feature correspondences [15], while others have chosen the dense approach in which all pixels in the image cast predictions about the output. In [16], the RGBD pixels perform joint prediction of 3D object coordinates and class labels for a subsequent RANSAC-based [17] optimization step to obtain the final object pose. Like many other works employing this approach, they utilize a variant of a Random Forest for casting the pixel-wise prediction due to the model's robustness and scalability to large dimensions and data quantities. A more recent variant dubbed *Latent-Class Hough Forest* [18] integrated the LINE-MOD feature into a Random Forest with a novel node split function to achieve state-of-the-art performance in occluded and cluttered environments.

More recently, there has been an increase in interest to deploy learning-based methods within various stages of a pose estimation pipeline. By employing supervised learning, [19] successfully train a CNN to learn discriminative feature descriptors which are matched using a k-NN algo-

rithm during inference time. In [14], the authors combined the local correspondence approach with deep learning by utilizing unsupervised deep auto-encoders for learning RGBD patch descriptors. The intuition is that under a suitable training procedure, the models will be able to learn even better features than hand-crafted ones such as LINE-MOD. The authors of [20] use a modular three-stage approach, in which they first perform instance segmentation to localize objects within the image, then regress surface coordinates with an encoder-decoder network and finally obtain the 6D pose using a geometric optimization algorithm. By combining feature learning with a geometric model-fitting algorithm, they achieve state-of-the-art performance on RGB as well as RGBD data.

Due to the challenges described in section 2.4, learning the 6D pose directly is an open research problem. In [21] a CNN is trained using a symmetry-aware pose loss to enforce distances in the feature space to be correlated with distances in the pose space, improving upon the work in [19]. The authors show that when a regression term is added, the learned features become more discriminative, but they achieve best pose inference by matching descriptors with a k-Nearest Neighbor algorithm rather than direct pose regression. Learning to estimate 6D object pose in a supervised manner can be tackled by classifying into a discretized pose space [12] or regressing key-points such as 3D bounding box corners [22]. More recently, researchers [5] have managed to regress object poses from RGB successfully by using the Lie algebra $\mathbf{so}(3)$ of the group of all rotation matrices SO(3) as the regression target. The deep neural network learns to regress the three $\mathbf{so}(3)$ parameters together with the depth component of the translation vector, while the other 2 components are estimated from the object's 2D bounding box.

In summary, a trend of learning-based methods gaining ground in the pose estimation area can certainly be identified. This work's primary contribution within this paradigm is adapting a state-of-the-art 2D object detection system, known as Faster R-CNN [1], for the 6D pose estimation task.

2.2 Faster R-CNN

Faster R-CNN is the state of the art 2D object detection system based on convolutional neural networks. The architecture is an evolution of the original R-CNN [23], which achieved best in-class performance at the time. The system employed a selective search [24] algorithm to create many object detection proposals in a given input image. Each detection proposal was re-shaped for the input size of a large CNN, and a forward pass through the network was computed. Instead of using a softmax classifier, the method employed a One-Versus-Rest SVM [25] classifier to obtain class scores. At the end of the pipeline, all regions with lower confidence than another region where the Intersection over Union (IoU) is higher than a learned threshold are rejected. The procedure is performed for all classes independently in order to prevent generation of multiple hypotheses per object instance. This technique is commonly referred to as non-maximum suppression.

The R-CNN architecture was subsequently improved upon in [26], resulting in higher object detection accuracy and an increase in speed by approximately 1 and 2 orders of magnitude for training and inference respectively. This improvement was achieved by sharing convolutional features for all detection proposals, which reduced the number of required forward passes through the CNN to just 1 per input image. Additionally, the multi-class SVM classifier was replaced by softmax, further simplifying the training procedure as well as increasing overall speed.

Finally, the Fast R-CNN system was improved upon further in [1] resulting in Faster R-CNN, which serves as the basis of this work. The authors achieved state-of-the-art object detection accuracy with an inference frame rate of 5fps, which represented a ten-fold improvement over Fast R-CNN. Faster R-CNN replaces the selective search algorithm by a Region Proposal Network (RPN) which exhibits significantly quicker runtime, thereby eliminating regional proposals as the computational bottleneck of the system. This architecture, as illustrated in Figure 2.2, consists of three main stages.

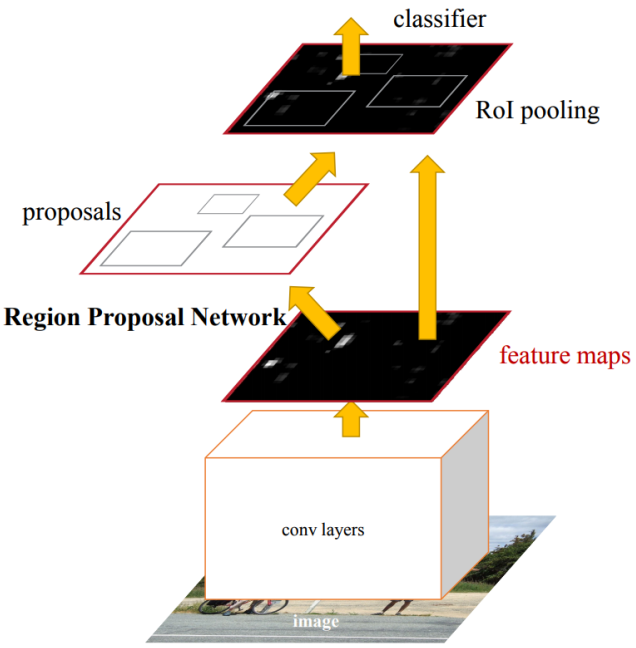


Figure 2.1: Faster R-CNN. Source: [1].

2.2.1 First Stage: Feature Extractor

The first part of the system is a convolutional network. In the original paper, the authors use a version of ZF net [27] which has 5 convolutional layers and 3 fully-connected ones, as well as a publicly available model of VGG16 [28], which has 13 convolutional layers and 3 fully-connected ones. The network parameters have been pre-trained on massive image classification datasets and subsequently fine-tuned to improve performance on the new task.

The purpose of this convolutional neural network is to serve as a feature extractor. The fully-connected layers that typically comprise the last stage of a network are removed due to their specialization for the original task, and the layer of interest is the last shared convolutional layer since it preserves the spatial structure of the image which is necessary for localization, and its features are generic enough for different tasks.

2.2.2 Second Stage: Regional Proposal Network

The Region Proposal Network is a smaller network that generates object detection region proposals. The network slides an $n \times n$ window on top of the convolutional feature map and

extracts a lower-dimensional feature vector at each location, which is subsequently used to evaluate region proposals at that location.

The regions are parametrized with respect to *anchors* that are centered at the given position within the feature map. Anchors have varying sizes and aspect ratios - this provides an efficient way to enable multi-scale object detection without the necessity to scale the input image or the dimensions of the proposal network. Each proposal has:

1. An associated *objectness* score, corresponding to the probability of the presence of an object within the given proposal. This way, the network learns to distinguish between background and foreground, allowing it to focus further computation where it is needed.
2. Regressed bounding box coordinates in terms of offsets with respect to the region's anchor. This allows the network to further refine the location of the proposed regions.

2.2.3 Third Stage: RoI Pooling and Regression

The purpose of the RoI (Region of Interest) pooling layer is to extract a fixed-size feature map for a given region proposal within the larger convolutional feature map. The layer performs max-pooling over a window defined by 2 corners in the convolutional layer, and the pooling size is approximately $W/w \times H/h$, where (W, H) and (w, h) are the dimensions of the extracted feature map and the output window, respectively. The main advantage of RoI pooling is that dimensions of the window of interest don't have to be divisible by the output dimensions - in case of a mismatch, the pooling size will simply be non-uniform.

Pooling is performed independently over all channels in the convolutional feature map, and the pooled features are fed into the final fully connected layers of the network. These are used to predict class scores by the softmax classification layer of the network, as well as final bounding box coordinates.

2.3 Faster R-CNN for 6D Pose Estimation

The motivation for employing Faster R-CNN for the task of 6D Pose Estimation stems partially from its high success in the related task of object detection. Detecting an object is a necessary prerequisite for estimating its pose, and the modular architecture of Faster R-CNN makes it possible to keep the detection part of the pipeline untouched. Conceptually, modifying the system to meet the first project milestone is not complicated, as the input data is also an RGB image, the type for which the original system was designed. The bounding box regression head of the original Faster R-CNN must be augmented with a pose estimation head, which will either regress or classify into a quantized pose space.

Another motivating factor for modifying Faster R-CNN are other successful efforts in adapting the system for different tasks and domains. By utilizing the idea of building a new head on top of the (albeit slightly modified) RoI layer as outlined above, [29] have successfully adapted Faster R-CNN for pixel-accurate instance segmentation, labeling the new system Mask R-CNN. In another relevant example, an R-CNN-based network for human pose estimation (by means of keypoint regression) and subsequent action detection [30] has been trained successfully, achieving state-of-the-art performance within the domain.

The intended contribution of this work is based on adapting Faster R-CNN for 6D Object Pose Estimation, and showing that its accuracy can be improved by utilizing the depth channel. The motivation to combine RGB and depth comes from the insight that these channels provide fundamentally different types of information - as stated in [3], in the LINE-MOD binary feature, color gradients provide useful edge information while normals extracted from the depth data provide a volumetric representation of objects, yet they are unreliable around edges. Due to their complementary nature, one can expect to increase the final accuracy by combining these channels over using any single one of them.

There are many possible ways in which depth data can be utilized to increase the accuracy of the pose estimation network. Conceptually, the most straightforward way would be integrating the depth channel into the same optimization procedure as color by simply treating it as

an equivalent fourth channel. However, this would necessitate retraining the entire network rather than just fine-tuning it. One alternative is to treat depth separately, utilizing it for pose refinement by applying a geometry-based optimization algorithm, similar to many of the related works [3, 16, 20]. Although this is a proven method for depth-based, pose refinement, it lacks a novelty factor as it has been implemented in many previous works. The approach taken in this work is re-purposing an RGB-pretrained feature extractor for the depth channel, keeping the pose estimation procedure an end-to-end process.

2.4 Challenges

2.4.1 Symmetry

Practical objects in 3D tend to exhibit varying levels of symmetry, which means that there exist several different views of the object that share the same visual appearance. The most extreme example of this phenomenon in 3D is an untextured sphere, which looks identical from all angles. The symmetrical nature of an object can be described by the set of its axes of symmetry. Formally, an object exhibits an *n-fold* axis of symmetry if its appearance is always identical under any rotation of $\frac{2\pi}{n}$ around the axis. For example, a 3D cube has the following set of axes of symmetry [31], as visualized in figure 2.2:

1. Three 4-fold axes that pass through the centers of opposing faces (red)
2. Four 3-fold axes that pass through the furthermost vertex pairs (green)
3. Six 2-fold axes that pass through the midpoints of opposing edges (blue)

Unlike this simple case, real-life objects that exhibit texture and complex shapes can be symmetric in non-trivial ways. The property of symmetry is a major source of pose ambiguity, and therefore a major conceptual challenge for pose estimation. It is desirable for indistinguishable poses to be treated as equivalent when designing, training and testing pose estimation systems.

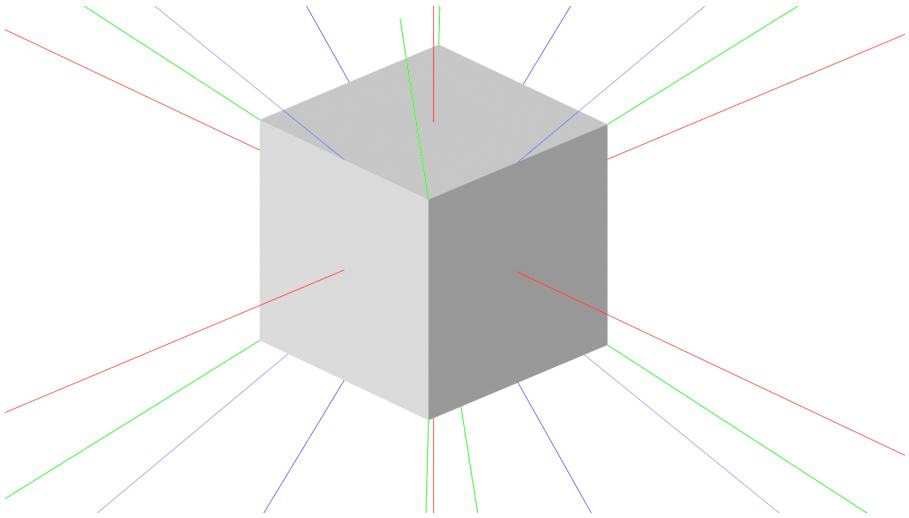


Figure 2.2: Axes of symmetry of a cube.

In practice, researchers have used various techniques to address the symmetry issue in training and testing. The authors of [21] decided to weigh the pose loss of a pose pair $(\mathbf{p}_1^{(i)}, \mathbf{p}_2^{(i)})$ by the term $\phi(\mathbf{p}_1^{(i)}, \mathbf{p}_2^{(i)}) = ||s(\mathbf{p}_1^{(i)}) - s(\mathbf{p}_2^{(i)})||_2^2$. Here, $s(\mathbf{p}_j^{(i)})$ is a rendered depth image of the object i at pose $\mathbf{p}_j$. This enforces that indistinguishable poses will be treated equally. In [12], different object views are treated as discrete classes. In order to take care of symmetry issues, the authors remove all training views that introduce pose ambiguity. In [22], a CNN is trained regress the 2D projections of 3D bounding boxes. To eliminate pose ambiguity, training is performed only on a limited view range, and another neural network is trained to classify into the correct pose range. In terms of evaluation, symmetry-aware formulations for pose error are commonly utilized, as described in section 2.6.

2.4.2 Occlusion

Occlusion occurs when an object of interest is not entirely visible due to the presence of other objects that are closer to the camera, as illustrated in figure 2.3.

This phenomenon presents a major challenge for pose estimation, especially when a significant portion of the object of interest is occluded. As mentioned in section 2.1, holistic approaches such as template matching are especially susceptible to occlusion, and therefore many works have relied on utilizing only local hypotheses to achieve robustness. However, recent findings

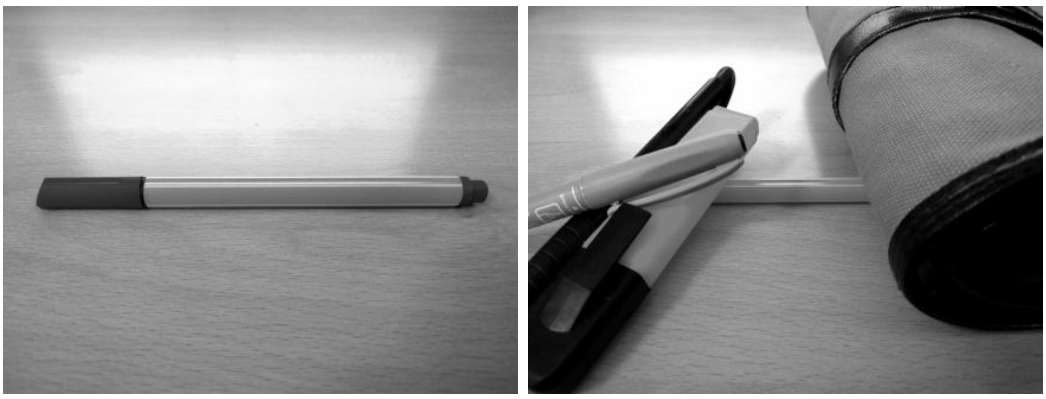


Figure 2.3: Example of occlusion as a source of pose ambiguity: in the right image, it is not clear whether the pen is oriented the same as on the left or whether the ends are flipped. Source: [2].

which are relevant for this work show that global reasoning based on a geometric check leads to highly accurate results [20, 32, 33]. These methods rely on checking geometric consistency in point clouds, either by utilizing available depth data or regressing surface coordinates. Overall, the depth channel offers great promise in terms of resolving occlusion due to its superior ability to discriminate between foreground and background.

2.4.3 Background Clutter

A high amount of background clutter makes it difficult to discern objects of interest, which presents a challenge for estimating their poses. Several of the works mentioned in section 2.1 [16, 18] address this issue explicitly.

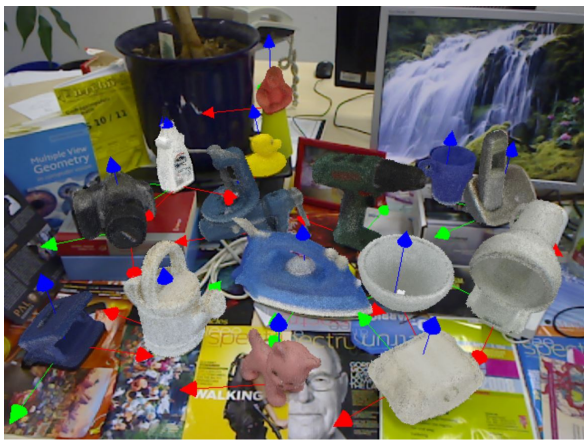


Figure 2.4: A highly cluttered scene containing 15 objects instances with ground truth pose projections. Source: [3].

Learning-based approaches to pose estimation [19, 21] propose achieving invariance to background influence by learning it. When synthetic training images are employed, it is trivial to create images with varying background but identical object poses, thereby forcing a model to learn representations which are invariant to background.

2.4.4 Texture

Texture provides rich discriminative features that are invariant to illumination, often disambiguating poses which would otherwise look identical. Therefore, textureless objects present a significant challenge when only RGB data is available, as patch-based methods fail and the object's overall shape must be considered. Textureless objects are of great interest since they are very common in industry, and lately have been subjected to increased focus [4]. When no discernible texture is present, the depth channel is especially useful, since the volumetric information that it contains is present regardless of the texture, and researchers have successfully tackled the challenge of textureless objects by utilizing RGBD data [34, 16, 3].

Figure 2.5: Textureless objects (left) with their ground truth pose projections (right). Source: [4].

2.5 Pose Regression

Formulating pose estimation as a regression problem must be done with great care, as picking unsuitable representations for the regression targets may render the training process difficult.

Due to its properties, the rotation component of the pose presents a greater difficulty than the object's translation. Regressing the 3×3 rotation matrix $\mathbf{R}$ directly is problematic, since the ground truth rotation matrices are limited to the 3D rotation group SO(3) such that:

$$\begin{aligned} \mathbf{R}^T\mathbf{R} = \mathbf{R}\mathbf{R}^T = \mathbf{I} \\ \det(\mathbf{R}) = 1 \end{aligned} \tag{2.1}$$

This form is highly parametrized and constrained, both of which are unfavorable properties in the context of learning. One could parametrize the matrix $\mathbf{R}$ by 3 Euler angles of rotation, but the challenge of regressing these directly is that they wrap around 2π, which makes a mapping difficult to learn for a neural network [5]. Ideally, the space of the regression targets would be unconstrained. One way to address this is to employ the associated Lie algebra $\mathbf{so}(3)$ of the rotation group SO(3) as the regression target, as done by the authors of Deep-6DPose [5]. This algebra consists of antisymmetric matrices with diagonal elements equal to 0, and therefore, it can be parametrized by a 3D vector as shown below:

$$\omega = \begin{bmatrix} \omega_1 \\ \omega_2 \\ \omega_3 \end{bmatrix} \rightarrow \widehat{\omega} = \begin{bmatrix} 0 & -\omega_3 & \omega_2 \\ \omega_3 & 0 & -\omega_1 \\ -\omega_2 & \omega_1 & 0 \end{bmatrix} \tag{2.2}$$

There exists a logarithmic map [35] that performs the transformation $\mathrm{SO}(3) \longrightarrow \mathbf{so}(3)$:

$$\widehat{\omega} = \frac{\phi}{2\sin(\phi)}(\mathbf{R} - \mathbf{R}^T) \tag{2.3}$$

Where the angle ϕ can be computed as:

$$\phi = \arccos \frac{tr(\mathbf{R}) - 1}{2} \tag{2.4}$$

This follows from properties of rotation matrices [36]. The corresponding inverse mapping $\mathbf{so}(3) \longrightarrow \mathrm{SO}(3)$, also referred to as the exponential map, is used to obtain the rotation matrix $\mathbf{R}$ from the Lie algebra given by ω:

$$\mathbf{R} = \mathbf{I} + \frac{\sin ||\omega||}{||\omega||}\widehat{\omega} + \frac{1 - \cos ||\omega||}{||\omega||^2}\widehat{\omega}^2 \tag{2.5}$$

2.6 Pose Error

Measuring the error of estimated 6D object poses is a non-trivial issue, and several approaches for defining this quantity have been proposed in the literature. Let us examine some of the common methods of quantifying pose error and discuss their merits:

2.6.1 Translation and Rotation Error

Given two poses $\hat{\mathbf{P}}, \bar{\mathbf{P}}$ with associated rotation matrices $\hat{\mathbf{R}}, \bar{\mathbf{R}}$ and translation vectors $\hat{\mathbf{t}}, \bar{\mathbf{t}}$, a simple way one can obtain a measure of the pose error is by calculating the translation and rotation error individually. Computing the error of the translation component is trivial:

$$e_{trans} = ||\hat{\mathbf{t}} - \bar{\mathbf{t}}||_2 \tag{2.6}$$

To quantify the error of the rotation, we utilize the following properties:

$$\mathbf{R}^{-1}(l, \theta) = \mathbf{R}(l, -\theta) \tag{2.7}$$

$$\mathrm{Tr}(\mathbf{R}(l, \theta)) = 1 + 2\cos\theta \tag{2.8}$$

Note again that (2.8) holds irrespective of the basis, following from the properties of SO(3) rotation matrices [36]. Hence, we arrive at:

$$e_{rot} = \arccos\left(\frac{tr(\hat{\mathbf{R}} \cdot \bar{\mathbf{R}}^{-1}) - 1}{2}\right) \tag{2.9}$$

The advantage of employing this error metric is that it provides an independent measure of accuracy for translation and rotation, as this decoupling can offer useful insight. On the other hand, this error metric does not take into account the object's model, and hence it isn't ambiguity-invariant.

2.6.2 Visible Surface Discrepancy

Visible Surface Discrepancy (VSD) is a pose estimation error measure that calculates the error over the visible part of the surface, thereby treating indistinguishable poses as equivalent. Given a predicted pose $\hat{\mathbf{P}}$, the ground truth pose $\bar{\mathbf{P}}$ of an object model $\mathcal{M}$ in an image I, the VSD is calculated as:

$$e_{VSD}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}, I, \delta, \tau) = \underset{p \in \hat{V} \cup \bar{V}}{\text{avg}}\ c(p, \hat{D}, \bar{D}, \tau) \tag{2.10}$$

$$c(p, \hat{D}, \bar{D}, \tau) = \begin{cases} 0 & \text{if } p \in \hat{V} \cap \bar{V} \wedge |\hat{D}(p) - \bar{D}(p)| < \tau \\ 1, & \text{otherwise} \end{cases} \tag{2.11}$$

As defined in [37]. Here, $\hat{V}$ and $\bar{V}$ are the 2D masks of the projections of $\mathcal{M}$ into the predicted and ground-truth poses, respectively. Since the pose is defined by the projection matrix in equation (1.1), the projection of the model $\mathcal{M}$ in pose $\mathbf{P}$ is simply equal to $\mathbf{P}\mathcal{M}$. The model $\mathcal{M}$ is described by a set of points in 3D space and a indices that define the triangle-based topology as is conventional in computer graphics. Furthermore, $\hat{D}$ and $\bar{D}$ correspond to the distance images of the projected images, and the average in equation (2.10) is performed over all pixels p in the union of the masks. The parameter δ is tolerance used for the estimation of the visibility masks, and τ represents the tolerance for pixel misalignment. Typical values of δ

and τ are $15mm$ and $20mm$, respectively [37].

2.6.3 Average Distance of Model Points

This error function, originally formulated by Hinterstoisser et al. [3] has two different formulations. For objects that do not have any indistinguishable poses, it is computed as:

$$e_{ADD}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \underset{x \in \mathcal{M}}{\text{avg}} ||\bar{\mathbf{P}}x - \hat{\mathbf{P}}x||_2 \tag{2.12}$$

Where $\mathcal{M}$ is the 3D model defining the object of interest. In case of objects where model $\mathcal{M}$ has some indistinguishable views, the formula is modified as follows:

$$e_{ADI}(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \underset{x_1 \in \mathcal{M}}{\text{avg}} \min_{x_2 \in \mathcal{M}} ||\bar{\mathbf{P}}x_1 - \hat{\mathbf{P}}x_2||_2 \tag{2.13}$$

However, even this formulation is not invariant under pose ambiguity since it considers points outside of the visible surface. Therefore, the VSD formulation of pose error is more desirable when symmetry is a major concern.

2.6.4 2D Intersection Over Union

Let $B_{box}(O)$ be the 2D bounding box of object O. Then, the intersection over union of the projections of model M under poses $\hat{\mathbf{P}}, \bar{\mathbf{P}}$ can be computed as:

$$IoU(\hat{\mathbf{P}}, \bar{\mathbf{P}}; \mathcal{M}) = \frac{area(B_{box}(\hat{\mathbf{P}}\mathcal{M}) \cap B_{box}(\bar{\mathbf{P}}\mathcal{M}))}{area(B_{box}(\hat{\mathbf{P}}\mathcal{M}) \cup B_{box}(\bar{\mathbf{P}}\mathcal{M}))} \tag{2.14}$$

The usual convention is to accept a pose if the IoU of the ground truth and predicted bounding boxes is higher than 0.5. While this provides a coarse measure of pose similarity, it is obviously not sensitive to fine differences in poses which are not captured by the 2D bounding box - for this purpose, different error metrics must be applied.

Chapter 3

Implementation

A successful execution of the project can be broken down into several major milestones that must be met in a certain order. At each step, there are several potential design choices that must be made regarding the tools and resources used, as well as approaches taken. This section describes the conceptual and practical choices made throughout the course of the project, as well as justification for the decisions taken.

3.1 Network Architecture

3.1.1 RGB Network

The architecture of the baseline RGB-only network is largely inspired by the Deep-6DPose network [5]. For reference, the Deep-6DPose architecture is provided in figure 3.1.

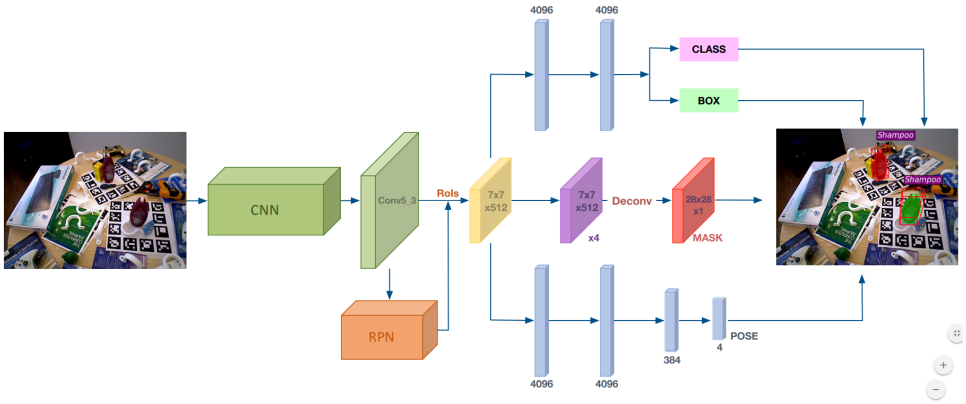


Figure 3.1: Architecture of Deep-6DPose. Note the semantic segmentation head that has been omitted in this work, as it serves no explicit purpose in the 6D pose estimation task. Additionally, the size of the last fully-connected hidden layer of the pose head has been increased from 384 to 2048 in this work, as this choice lead to better convergence in initial experiments. Source: [5].

It has all the components of Faster R-CNN: a feature extractor (the 13 convolutional layers of VGG16), the region proposal network (RPN) followed by an RoI pooling layer that feeds into the fully-connected regression heads. In parallel with the original classification and bounding box regression head, there is another one that regresses 4 parameters necessary to recover the 6D pose. The RoI-pooled features serve as input to the head consisting of 3 additional hidden layers as depicted in figure 3.2.

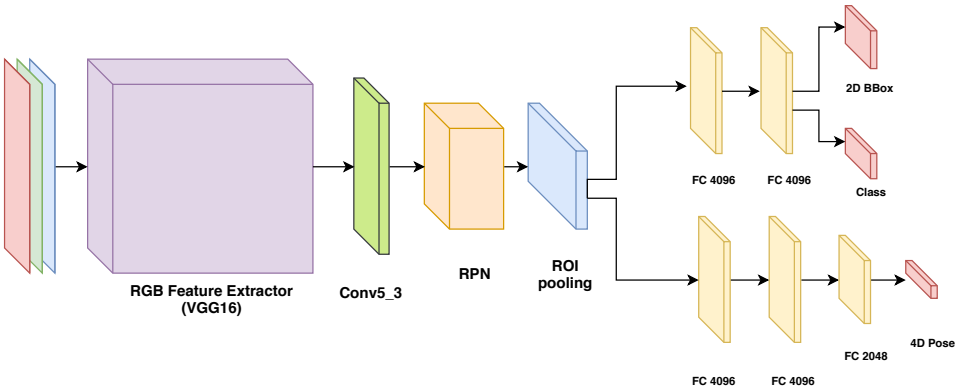


Figure 3.2: Architecture of the RGB-based 6D pose estimation deep neural network.

The 4-layer pose head regresses the 4-dimensional pose vector defined as:

$$p = (\omega_1, \omega_2, \omega_3, t_z)^T \tag{3.1}$$

Where ω_i is the i-th element of the Lie algebra vector ω as defined in equation (2.2). The additional hidden layer of the pose head is motivated by the higher complexity of its associated task. Given the regressed Lie algebra ω, the rotation matrix $\mathbf{R}$ is recovered using the exponential map in $\mathbf{so}(3)$, in accordance with equation (2.5). Using the equations below, the full translation vector is recovered under the assumption that the center of the 2D bounding box corresponds to the projection of the object's 3D center onto the image:

$$t_x = t_z \frac{u_x - c_x}{f_x} \tag{3.2}$$

$$t_y = t_z \frac{u_y - c_y}{f_y} \tag{3.3}$$

Where (u_x, u_y) are the center coordinates of the 2D bounding box and c_x, c_y, f_x, f_y are parameters of the intrinsic camera calibration matrix, which has the form:

$$\widehat{\omega} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

Specifically, (f_x, f_y) are the focal lengths and (c_x, c_y) are the coordinates of the principal point of the camera.

3.1.2 RGBD Network

The network that utilizes both RGB and depth inputs has an architecture that largely resembles that of the RGB one, with a couple of differences to allow employing both input modalities. In addition to the RGB feature extractor, there is a parallel network for extracting features from depth images. The features extracted from these two networks are concatenated, and subsequently used as input into the convolutional *Conv_merge* layer, which produces a feature layer that is equal in shape of its two input layers. The resulting layer is then fed to the rest of the network which is identical to the RGB-only architecture.

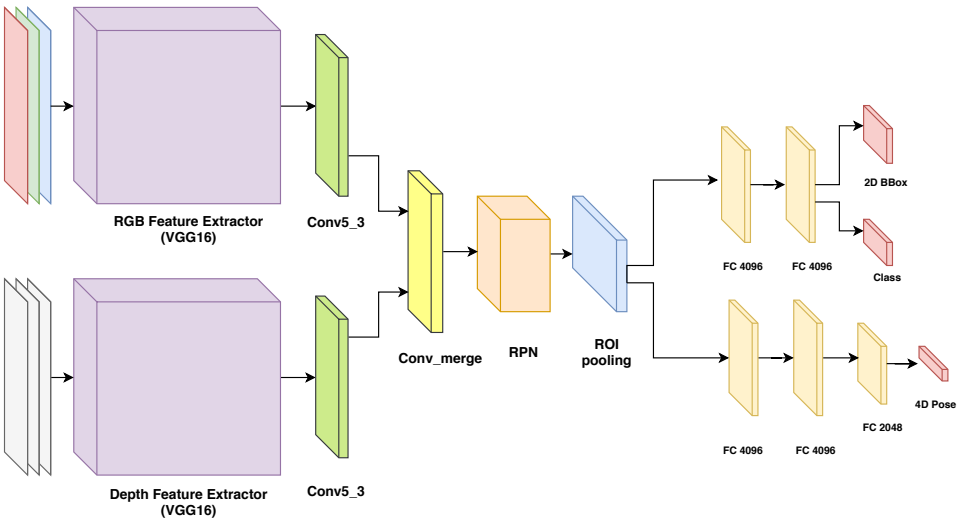


Figure 3.3: Architecture of the RGBD-based 6D pose estimation deep neural network.

In principle, the two feature-extracting networks could employ considerably different architectures, but for the purposes of this work, both are initialized to a pre-trained VGG16, as training such a deep network from scratch would require considerable time and compute resources. We hypothesize that although the filters of the VGG16 network have been optimized for color rather than depth channels, they are general enough for the extracted features to provide information that is not found in the RGB-derived features. If such features can be extracted, the higher network layers will adapt to utilize them during the training process, and moreover, the feature extractor itself will be fine-tuned further for the new data modality. Conceptually, this is an example of transfer learning [38], which is the general concept of knowledge transfer from one domain to another.

3.2 Framework

With the rising popularity of deep learning, several frameworks that enable fast creation and training of deep learning models have emerged. In general, these frameworks contain libraries of classes and methods that correspond to the various building blocks and functions employed in typical deep learning architectures. The raw computational aspects are typically implemented with support for GPU acceleration, exploiting the massively parallel nature of typical deep

learning workloads and providing a major speed-up over sequential CPU implementations. The major productivity boost of these frameworks stems from the fact that they provide high-level APIs to the optimized low-level routines, thereby enabling fast prototyping as well as excellent performance.

Some of the most popular deep learning frameworks as of June 2018 include Tensorflow [39], Caffe [40], Pytorch [41], Keras [42] and many others. The frameworks differ from each other in terms of their underlying computational models and supported API's for different programming languages, but they can all be utilized to reach the same goal, and the choice of a particular one is largely down to user preference. As such, Pytorch was chosen as the framework for this project.

Pytorch is a deep learning framework that was designed specifically for Python. In addition to GPU-accelerated tensor computation, support for various layers, dataset processing and several optimization procedures, it features a novel library for automatic differentiation of variables with respect to differentiable Tensor operations. Since contemporary methods of optimizing deep learning models rely on gradient computation, this is extremely useful in practice. Furthermore, as described in the next section, there are several open-source implementations of Faster R-CNN in Pytorch which can be utilized as the starting point for implementing the 6D pose estimation extension of the system.

3.3 Computational Resources

Inference and training of large-scale deep learning models is generally a computationally intensive task. The specific requirements are largely determined by the choice of the feature extractor, which contains the majority the network's parameters. The amount of required GPU memory starts at 3GB [43] for a 16-layer configuration of VGG (known as VGG16), which is the extractor utilized in this work. Given such requirements, the computationally demanding experimental aspects of the project have been run on high-performance remote machines provided by Amazon Web Services (AWS).

	GPUs	GPU Memory	GPU Model	vCPUs	Price/hr
p2.xlarge	1	12 GB		4	$0.900
p2.8xlarge	8	96 GB	Tesla K80 (2014)	32	$7.200
p2.16xlarge	16	192 GB		64	$14.400
g3.4xlarge	1	8 GB		8	$1.14
g3.8xlarge	2	16 GB	Tesla M60 (2015)	16	$2.28
g3.16xlarge	4	32 GB		32	$4.56
p3.2xlarge	1	16 GB		16	$3.06
p3.8xlarge	4	64 GB	Tesla V100 (2017)	32	$12.24
p3.16xlarge	8	128 GB		64	$24.48

Table 3.1: AWS EC2 instance comparison.

AWS encompasses a set of cloud computing solutions, including the Elastic Compute (EC2) service which is of particular interest for this project. This service provides virtual computing environments, known as *instances*, which can be accessed remotely through the SSH protocol. The instances are highly configurable in terms of CPU, GPU, storage as well as software. Table 3.3 outlines different configurations of instances that are recommended for deep learning, together with their respective pricing. For the purposes of this project, the single-GPU instances provide sufficient computational resources at an affordable price point. During initial experiments, the **g3.4xlarge** instance has shown to be sufficient for running the Pytorch Faster R-CNN implementation of [43]. Under the (now tested) assumption that computational requirements would not rise dramatically due to addition of RGBD pose estimation into the system, this instance choice has been kept for the duration of the project.

3.4 Dataset

The primary purpose of the dataset for this project is to demonstrate the feasibility of adapting Faster R-CNN for object pose estimation and facilitate the RGB vs RGBD comparison, rather than addressing a specific challenge in-depth. As such, the dataset from Tejani et al. [18] was chosen due to moderate amount of occlusion and clutter, as well as the small amount of object classes which are either textured or textureless and exhibit varying degrees of symmetry.

This RGBD dataset contains 6 image sequences (one for each of the objects depicted in figure 3.4) with a total of 2067 images, as well as rendered training images. It is publicly available on

Figure 3.4: Object renders from Tejani et al. The corresponding object classes (starting from top left) are *Camera*, *Coffee*, *Joystick*, *Juice*, *Milk* and *Shampoo*.

the 6D Pose Estimation challenge website[1], and it comes with CAD objects models, intrinsic camera parameters as well as Python rendering scripts. Figure 3.5 shows example color and depth images from the image sequences.

3.5 Code

3.5.1 Starting Point

The original implementation [1] of Faster R-CNN is publicly available in MATLAB. However, since its release, the system has been reimplemented in many other frameworks under permissive licenses. For the purposes of this project, open source Pytorch implementations of Faster R-CNN are of interest as the basis for the 6D pose estimation extension.

One such implementation, called Simple Faster R-CNN [43] has been released recently under the MIT License. It is a well-documented, minimal implementation that can be run purely Python, and its object detection accuracy is slightly higher than the original version, achieving 0.712 mAP versus 0.699 mAP of [1] (mAP, short for mean average precision, is a common measure of 2D object detection accuracy). The code is composed of the three main components listed

[1] `http://cmp.felk.cvut.cz/sixd/challenge_2017`

Figure 3.5: Example RGB and corresponding depth images from the Tejani et al. dataset. The testing images typically contain 2 or 3 instances of the same class. Note that the depth images usually exhibit relatively high levels of noise.

below, reflecting Faster R-CNN's modular architecture:

1. *Feature extractor:* The deep neural network that extracts a convolutional feature map from the input image. By default, this is VGG16.

2. *Region Proposal Network:* The neural network that scores object detection proposals, as described in section 2.2.2.

3. *Classification and regression head:* The layer which takes RoI features as input and outputs a probability distribution over all classes as well as the regressed bounding box coordinates, as described in section 2.2.3.

As a consequence of this modular design, making required architectural changes in the code is greatly simplified. For example, when the Faster R-CNN is extended for 6D pose estimation

using RGB data, the required addition of the pose regression head does not influence the rest of the network's code.

3.5.2 System Architecture

The system design of the finished pose estimation network follows that of Simple Faster R-CNN, with numerous additions and adaptations. Figure 3.6 depicts a high-level diagram of the complete software architecture, including the main modules and the relationships between them.

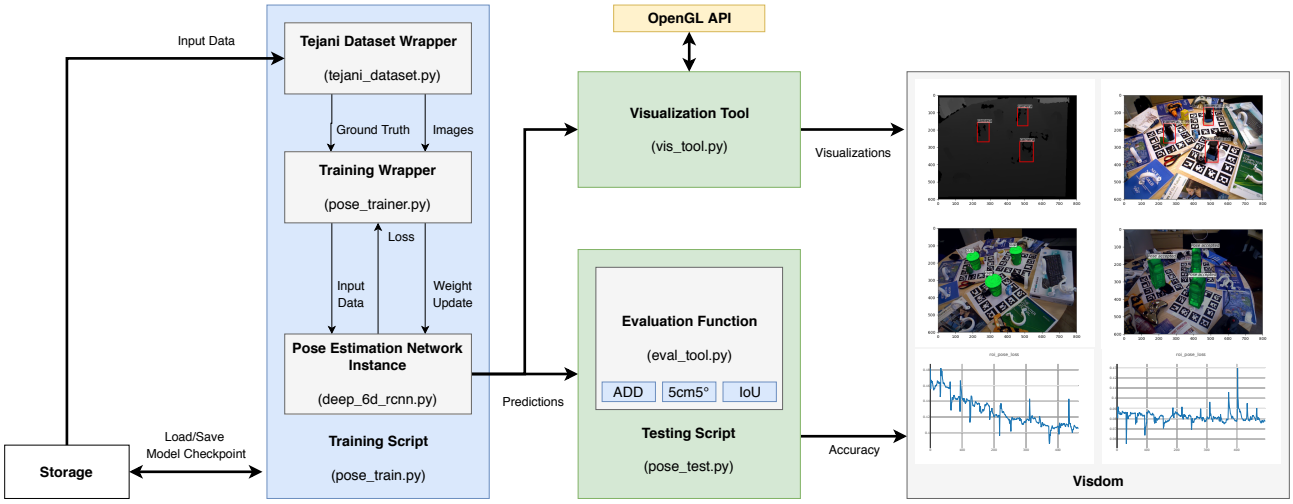


Figure 3.6: System architecture diagram.

The next several sections examine the individual system components in closer detail. For the sake of completeness, the complete code is publicly available on github [2].

Dataset Wrapper

The Tejani et al. dataset wrapper serves as the front-end for the rest of the system, replacing the VOC dataset wrapper in the original implementation. At instance creation time, the constructor requires the dataset path and split (either *trainval* or *test*). Subsequently, it performs

[2] `https://github.com/pufik1337/fyp/tree/rgbd`

the random data split and pre-computes the mean and standard deviation statistics over the training data, which are later used to normalize pose vectors.

The main function of interest is the **get_example** method, which fetches the i-th RGB and (optionally) depth image, converts them into array format and packs them with the associated class label, bounding box and pose annotations. Since the pose regression targets have the 4D format given by equation (3.1), for a each example, the wrapper must perform the $SO(3) \longrightarrow \mathbf{so}(3)$ transformation. Before being returned, the elements of the pose vector are normalized using pre-computed statistics. The second relevant function of the dataset wrapper is the **shuffle** method, which permutes the id's of training examples. Care is taken so that the data split is unaffected by this step.

The Model

The overall network architecture is defined in the *deep_6d_rcnn.py* file as the **Deep6DRCNNVGG16** class, extending the **FasterRCNN** class. At construction time, an instance of the VGG16 network is initialized to be used as the feature extractor, with the option to use the parameters of a pre-trained model. The RPN network and ROI pooling stages are unchanged with respect to the description in sections 2.2.2 and 2.2.3.

In terms of RPN anchor parameters, this work uses 5 different scales of $(16, 32, 64, 128, 256)$ pixels and 3 aspect ratios of $(1:2, 1:1, 2:1)$. Since all scale and aspect ratio combinations are considered, this means that the RPN evaluates 15 region proposals at each position in the convolutional feature map, enabling object detection at many different scales.

The pose regression head is wrapped in a distinct class **VGG16PoseHead**, instantiating the final layers depicted in the architecture diagram in figure 3.3. Each of the major network modules implements its own **forward** method, which defines how data is propagated through the network in a forward pass.

For purposes of testing, the **Deep6DRCNNVGG16** class implements the **predict** method which returns predictions of object classes, poses and 2D bounding box locations in image

coordinates. The majority of predictions is filtered out based on a confidence score threshold of 0.7 and non-maximum suppression with an IoU threshold of 0.3.

The Training Wrapper

The file *pose_trainer.py* contains the **Deep6DRCNNPoseTrainer** class, which wraps around the model that is being trained and contains functions that aid the training process. Its members include the hyperparameters of the learning process, an instance of the model's optimizer (SGD with momentum, as detailed in the next section) from the *torch.optim* package, as well as instances of **ProposalTargetCreator** and **AnchorTargetCreator**, which are needed to associate individual region proposals with ground truth quantities.

Training and Testing Script

The script *pose_train.py* is the top-level file during the training procedure, instantiating the dataset wrapper, the trainer as well as the network itself. When the script is launched, a configuration is loaded from the configuration file *config.py*, where several parameters of the training procedures (number of epochs, file paths, hyperparameters, etc.) are specified. Instead of initializing the entire network with random weights, it offers the option to pre-load network parameters from a previously saved model file (a checkpoint).

The script loops over the training data for the specified number of epochs. For every mini-batch of 1 training example, it performs a forward pass to calculate the loss function and a backward pass to obtain its gradient and perform an update. In a pre-specified interval of mini-batches, the loss function is plotted and predictions are visualized.

After each epoch, the network accuracy is evaluated by the evaluation function **eval**, which implements several of the pose error metrics described in section 2.6, as well as mAP (mean Average Precision) for evaluating 2D object detection accuracy. All evaluated poses are transformed using the exponential map $\mathbf{so}(3) \longrightarrow SO(3)$ and equations (3.2), (3.3) to obtain the full rotation matrix $\mathbf{R}$ and translation vector t. The testing script *pose_test.py* only implements the

evaluation functionality, leaving out the training epochs. Once training is finished, the latest checkpoint is saved to storage so that it can be re-used for further training or inference.

Visualization

The visualization tool uses Visdom [44] as a backbone to enable live visualization of training and evaluation. Since all experiments run on remote machines, producing live visualizations can be problematic. Visdom overcomes this by sending the visualizations to a server such that they can be accessed using a web browser. The tool sends data to a port of the remote machine, which is accessed from a local machine using *ssh port forwarding*, which employs the ssh protocol to forward data from the remote port to a local port (both the remote and local machines use port 8097 in our case, although the choice is arbitrary).

The visualizations include graphs of the loss function against time, as well as bounding box and pose predictions. To create pose visualization, the poses are projected using the rendering scripts from the SIXD Toolkit [45] and subsequently overlaid on top of corresponding images. The rendering scripts make use of the glumpy library [46], which provides an abstraction layer around OpenGL. All additional 2D overlay (such as labels and boxes) is drawn using Matplotlib [47]. Several examples of these visualizations are provided in Chapter 5.

3.6 Initialization

Other than the feature extractor networks, which are initialized with weights from a pre-trained VGG16[3] network, the rest of the network is initialized randomly. Weights are drawn from a zero-mean Gaussian distribution with standard deviation 0.01 or 0.001, depending on the layer size. Since the sum of Gaussian variables has variance equal to the sum of the individual variances, wider layers are initialized with narrower Gaussian distributions to keep the output activation variance under control. In related work, an optimal analytic solution to the initialization

[3] https://s3-us-west-2.amazonaws.com/jcjohns-models/vgg16-00b39a1b.pth

problem has been derived [48], although this is mostly of interest for very deep networks (with 30 or more layers).

Chapter 4

Training

4.1 Loss Function

Within the context of optimization in deep learning, the purpose of the loss function is to map the predictions of a model to a scalar quantity that can be subsequently minimized during training in order to achieve higher prediction accuracy. Like Faster R-CNN and other similar systems, our network consists of (i) an object detection and pose estimation network and (ii) a region proposal network. Both of the individual nets must be trained in order for the system to function properly, hence a loss function must be defined for each network.

4.1.1 Pose Estimation Network

The loss of the pose estimation network is defined on randomly sampled regions of interest. A region is considered to be positive if its IoU with a ground truth bounding box exceeds 0.5 while anything below this threshold is considered negative. In this work, the the total loss of the pose network consists of a weighted sum of three individual loss terms:

$$L = \alpha_1 L_{cls} + \alpha_2 L_{loc} + \alpha_3 L_{Pose} \tag{4.1}$$

Here, L_{cls} and L_{loc} are defined as in the original Faster R-CNN paper [1] - a cross-entropy loss for the predicted class distribution, and a smooth L1 loss for the regressed bounding box coordinates. For each training example, 2000 RoI's are sampled such that the fraction of foreground (positive) to background (negative) regions is 0.25. Note that the localization and pose loss is defined only for positive regions, and equals 0 for negative regions. The per-example classification cross-entropy loss is calculated as:

$$L_{cls}(f, y) = \frac{1}{N_p + N_n} \sum_{i=1}^{N_p+N_n} -\log\left(\frac{e^{f_{y_i}}}{\sum_{j=1}^{M} e^{f_{j,i}}}\right) \tag{4.2}$$

Here, N_p and N_n is the number of positive and negative RoI's respectively, $f_{j,i}$ is the j-th element of the output vector and y_i is the index of the true class label in the i-th region of interest. The localization loss term L_{loc} is defined as:

$$L_{loc}(t, v, y) = \frac{1}{N_p} \sum_{i=1}^{N_p} smooth_{L_1}(t_i^{y_i}, v_i) \tag{4.3}$$

$$smooth_{L_1}(x, y) = \begin{cases} 0.5(x-y)^2 & \text{if } |x-y| < 1 \\ |x-y| - 0.5, & \text{otherwise} \end{cases} \tag{4.4}$$

where $t_i^{y_i}$ is the regressed bounding box (parametrized by corner coordinates) corresponding to the ground truth class y_i and v_i is the ground truth bounding box of the i-th positive region of interest. Finally, the pose loss is defined as follows:

$$L_{Pose} = \frac{1}{N_p} \sum_{i=1}^{N_p} |\hat{\omega}_i - \bar{\omega}_i| + \beta|\hat{t}_{z,i} - \bar{t}_{z,i}| \tag{4.5}$$

where $\hat{\omega}_i$ and $\bar{\omega}_i$ is the Lie algebra $\mathbf{so}(3)$ of the ground truth and predicted rotation matrix respectively, while $\hat{t}_{z,i}$ and $\bar{t}_{z,i}$ are the corresponding depth components of the translation vectors in the i-th positive region of interest. The parameter β controls the relative weights of the rotation and depth components.

4.1.2 Region Proposal Network

Similarly to [1], the RPN is trained using the sum of two loss terms. The first is the RPN classification loss in accordance with equation (4.2) that classifies regions into foreground and background. The second term is the smooth L1 localization loss (4.4) of the RPN bounding box proposals. In this case, the loss is calculated over 256 sampled regions with a positive-to-negative ratio of 0.5. Positive regions correspond to an IoU of 0.7 with a ground truth bounding box while negative regions must have IoU lower than 0.3. Only the positive regions factor into the localization loss, and regions which are neither positive or negative do not contribute into RPN training.

4.1.3 Joint Approximate Training

An important issue in training the entire system is choosing an approach that trains both of the subnetworks in a way that favors minimizing computation and maximizing accuracy. In principle, the pose estimation network and the RPN could be trained independently, but this would defeat the purpose of the computation-saving feature sharing between the networks. In this work, we adopt *joint approximate training*. Compared to *alternating training*, which only trains one network while completely the other one, it has been empirically shown to be the faster by 25% to 50% without carrying an accuracy penalty [49].

In this method, region proposals from the RPN are treated as fixed inputs to the regression heads during a single forward pass, and the total loss of the network consists of a sum of the RPN and pose estimation network losses. The gradient signals from the networks combine at the last convolutional layer of the feature extractor, where their sum propagates to the very back of the network. This approach is considered approximate because the gradient of the bounding box coordinates with respect to the RPN proposals is ignored even though it is a function of the network's parameters [1].

4.2 Initialization

4.3 Data Split

Following the training procedure of [5], the test sequences of the Tejani et al. dataset are split such that 30% of the data is used for training and 70% is reserved for testing. Finding a suitable ratio is a matter of balancing two competing goals: achieving good convergence on training data and low variance on the test data, and the above ratio is a common heuristic choice in this dilemma. Since the 6 sequences have different lengths, the training examples are sampled from each sequence individually using this ratio. Table 4.1 show the image counts in the individual sequences for the resulting training and testing sets.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Total
Training Images	80	124	163	123	28	102	**620**
Testing Images	185	290	380	287	67	238	**1447**

Table 4.1: Image counts in the training and testing sequences of the Tejani et al. dataset.

4.4 Optimization Procedure

The overwhelming majority of successful methods are methods based on gradient descent, which relies on computing the analytic gradient of the loss function with respect to the network's parameters and performing an update step in the direction of the anti-gradient.

4.4.1 Stochastic Gradient Descent (SGD)

One of the simplest methods of updating the network weights W_t in iteration t is to perform a fixed-sized step along the anti-gradient of the loss function:

$$W_{t+1} = W_t - \eta \nabla_{W_t} L(W_t, X_t, y_t) \tag{4.6}$$

where η, usually referred to as the learning rate, is a parameter controlling the step size along the anti-gradient. When the gradient is computed over a mini-batch of data, as opposed to the entire training dataset, the gradient descent method is considered stochastic. In practice, SGD achieves good convergence properties while being much more computationally tractable compared to standard gradient descent [50].

4.4.2 SGD with Momentum

In practice, it is often found that performing a step given by the weighted average of the gradient and the previous step yields better convergence [51]. This is analogous to attaining momentum while moving through the landscape of the loss function, hence this update method is referred to as SGD with momentum. Mathematically, this can expressed as:

$$W_{t+1} = W_t - \eta V_t \tag{4.7}$$

$$V_t = \beta V_{t-1} + (1 - \beta) \cdot \nabla_{W_t} L(W_t, X_t, y_t) \tag{4.8}$$

Where β is the momentum hyperparameter (note that for $\beta = 0$, this reduces to standard SGD). Given the success of SGD with momentum in training Faster R-CNN [1] and its derivative architectures [5, 29], this optimization method was chosen for training the pose estimation networks in this work.

4.4.3 Hyperparameter Choices

The values of hyperparameters α_1 - α_3 and β have a direct effect on the speed and outcome of the optimization procedure. Intuitively, the values of these parameters set the relative importance of the individual terms of the loss function - by controlling their magnitudes, they control each term's contribution to the gradient.

The baseline inspiration [5] mentions no details about their methodology of the selection of these parameters. Since their work utilizes an additional loss term for semantic segmentation, re-using the same values for the other terms without justification would unlikely result in an optimal choice. A common approach that was considered is to reserve a part of the data for a *validation* set, separate from the training and test sets, and choose parameters based on the calculated error on this set. In initial experiments, it was found that using this approach:

- The pose loss converges with much more difficulty compared to the other loss terms
- The training and validation pose error are highly correlated

Therefore, final hyperparameters were chosen based on training accuracy, as the network was discovered to be much likelier to underfit rather than overfit data, likely due to the inherent difficulty of the pose estimation problem. The empirically determined parameter values are $\alpha_1 = \alpha_2 = 1.0$, $\alpha_3 = 4.0$, $\beta = 1.5$.

4.4.4 Input Data

As outlined in section 3.5, all of the 4D training regression targets are normalized to have 0 mean and unit variance. The transformation is handled by the Python dataset wrapper, which pre-computes both statistics μ_{train} and σ_{train} on the entire training dataset.

During testing time, the regressed Lie algebra and depth component of the translation are transformed using $p \rightarrow p \cdot \sigma_{train} + \mu_{train}$ before the full 6D Pose is recovered using equations (3.2) (3.3) and (2.5).

The size of all input images is 640×480. Since the VGG feature extractor expects inputs in the range 0-255, the depth images are mapped to this range by the Tejani at al. dataset wrapper. Each depth image is stacked vertically 3 times to fit the input dimensions of the convolutional filters in the VGG feature extractor. All data (images, class labels, bounding boxes and poses) returned by the wrapper is in numpy array [52] format.

4.4.5 Training Procedure

The training procedure diverges from that of [5] with the intention to reduce training time significantly while still attaining good accuracy. The network has been trained for 55 epochs of 620 iterations each, resulting in $34.1k$ iterations in total. Before each training epoch, the training sequences are shuffled to randomize the image ordering in the epoch. For sake of comparison, note that Deep6D-Pose was trained for $350k$ iterations, which would be prohibitive given this project's time and compute constraints.

In each iteration, a mini-batch of 1 training example is propagated through the network, and the parameters are updated in accordance with the optimization procedure described in section 4.4. In each iteration, 2000 RoI's are sampled with a 1:3 ratio of foreground to background regions, and the multi-task loss (4.1) is obtained from this sample. Analogously, the RPN loss is obtained as described in section 4.1.2. The learning rate η was initialized to $1e^{-3}$, and further decreased to $1e^{-4}$ after the 9th epoch. Subsequently, the learning rate was halved on epochs 24 and 39. In the final 15 epochs, the learning rate was multiplied by 0.9 after each epoch as initial experiments showed the loss saturating more quickly without further reduction of the learning rate. To ensure a fair comparison between the RGB and RGBD networks, both networks were trained using the same procedure and data split (the dataset wrapper was initialized with the same random seed). Figures 4.1 and 4.2 show how the pose loss and test accuracy evolve during the training procedure for the RGBD and RGB networks respectively.

It can be seen in both figures that as soon as the training loss starts saturating, so does test accuracy. Interestingly, the most profound surge in accuracy occurs around epoch 10 - shortly after the first decrease in the learning rate - while the following decreases yield increasingly diminishing returns. Overall, the RGBD network converges faster despite the initial instability in the training loss, and achieves higher accuracy in the plotted ADD metric. The accuracy exhibits low variance in the final stages of the training process - over the last 10 epochs, the mean of the difference in test accuracy is approximately 0.04 while its standard deviation is approximately 0.008. Therefore, it is very unlikely (5σ, a chance of 1 in 1744278) that the performance difference can be attributed to noise in the training procedure. At the end of the

training procedure, checkpoints of both models were saved and further evaluated on accuracy metrics, as detailed in the next chapter.

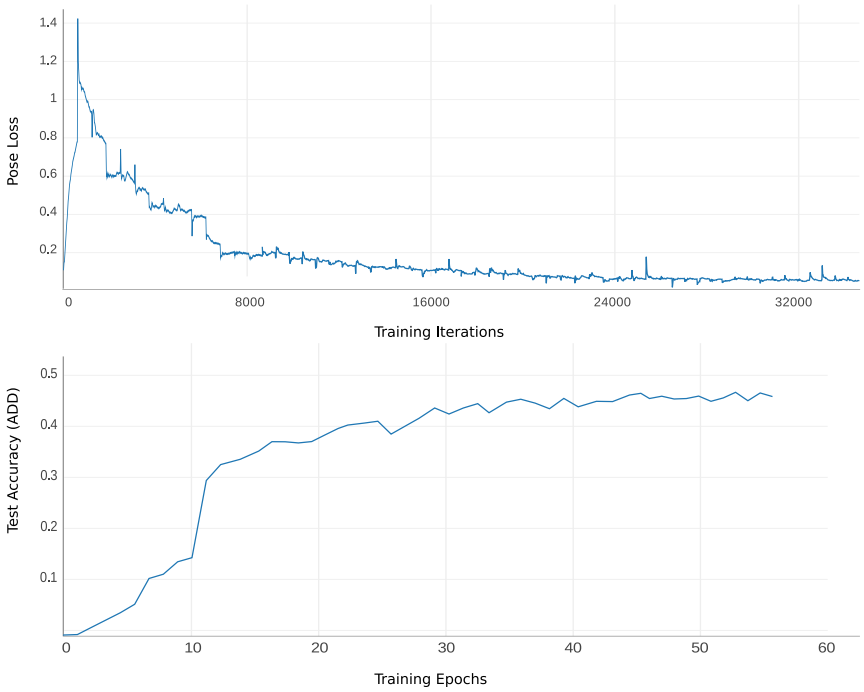


Figure 4.1: RGBD network pose loss and ADD test accuracy as a function of training iterations.

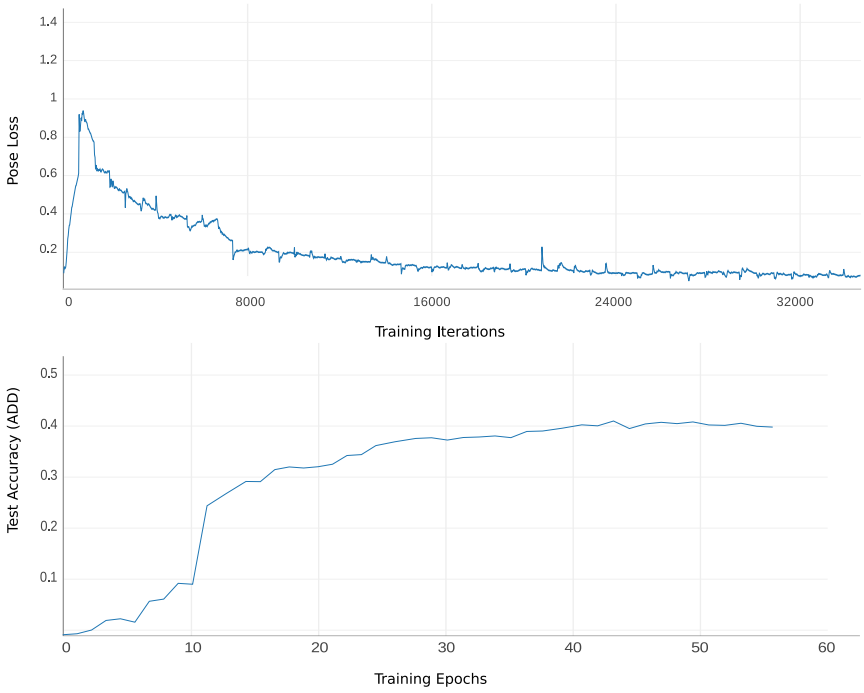


Figure 4.2: RGB network pose loss and ADD test accuracy as a function of training iterations.

Chapter 5

Evaluation

5.1 Test Accuracy

In this section, the accuracy of both the RGB and RGBD networks is evaluated on several standard accuracy metrics, as done in works such as [22] and [53] and subsequently compared to Deep6DPose [5], which represents the state-of-the-art in 6D object pose estimation on the Tejani et al. dataset without use of synthetic data for training. The test results are subsequently analyzed and discussed.

5.1.1 2D Object Detection mAP

The accuracy on the 2D object detection is of interest in this work, as the predicted bounding boxes are used (in conjunction with the predicted depth) to compute the translation component of the 6D pose. To evaluate the networks on this task, we use the standard AP (Average Precision) metric as defined in the PASCAL Visual Object Classes (VOC) Challenge [54]. This quantity is calculated for a list of bounding box location predictions with associated confidence values and ground truth coordinates. In order to define AP, two terms need to be introduced:

- *Precision* : The percentage of correct predictions, which is penalized by false positives.

- *Recall* : The percentage of all ground truth positives that have an associated positive prediction, which is penalized by false negatives.

For a given prediction to be positive, its associated confidence must exceed a threshold t_{pos}. A given positive prediction is considered a true positive if its IoU with a ground truth bounding box exceeds a different threshold t_{IoU}. Both precision and recall are a function of the confidence threshold t_{pos}, and the precision-recall curve is defined by the relationship of the two quantities as t_{pos} varies. In this work, the Average Precision is computed in accordance to the VOC Challenge definition by averaging the 11 precision values obtained when varying recall in discrete steps of 0.1 from 0.0 to 1.0. The quantity is computed for each class separately, and the mean Average Precision (mAP) is defined as the average AP over all classes.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	mAP
RGB Network	100.0	100.0	100.0	100.0	100.0	90.9	98.5
RGBD Network	100.0	100.0	100.0	100.0	100.0	90.9	98.5
Deep-6DPose [5]	99.8	100.0	99.8	99.2	99.7	99.5	99.6

Table 5.1: 2D object detection test accuracy on the Tejani et al. dataset: **AP**.

As can be seen in table 5.1, both networks achieve almost perfect mAP, and the only sequence with failure cases is the *Shampoo*. In comparison, Deep-6DPose achieves 100% on the *Coffee* sequence and over 99% on the other ones.

Figure 5.1: Example testing image from Tejani et al. with ground truth (left) and predicted bounding boxes (right). The confidence scores are shown above corresponding bounding boxes.

5.1.2 ADD metric

The 6D pose estimation accuracy of both networks has also been compared using the ADD metric. The average distance of model points is computed in accordance with equation (2.12), and only predicted poses which result in the average distance being less than 10% of the object's diameter are accepted. Table 5.2 lists the accuracy on each of the 6 test sequences as well as the average accuracy over all sequences.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Average
RGB Network	27.9	22.0	15.9	66.7	55.3	60.1	41.3
RGBD Network	36.8	24.5	13.3	77.0	52.5	67.9	45.3
Deep-6DPose [5]	80.4	35.4	27.5	81.2	71.6	75.8	62.0

Table 5.2: 6D pose estimation test accuracy on the Tejani et al. dataset: **ADD**.

Figure 5.2: Example success cases of the RGBD Network pose predictions under the **ADD** metric. The ground truth poses (left) and the predictions (right) are projected using the object poses and overlaid on top of the corresponding test images.

It can be seen that the RGBD network significantly outperforms the RGB counterpart on the *Camera*, *Juice* and *Shampoo* sequences, while performing slightly better on the *Coffee* sequence and slightly worse on the *Joystick* and *Milk* sequences. On average, there is a notable but not dramatic improvement of 4%. The RGBD network lags 16.7% behind the state-of-the-art, which seems reasonable given that it has been trained for less than 10% of the time. Examples of success cases on the *Shampoo* and *Juice* sequences, which were the most successful ones for the RGBD network, are visualized in figure 5.2.

In some cases, as shown in figure 5.3 the network makes errors of varying severity. In case of the mis-predicted joystick pose, the culprit is a small misalignment, while in case of the milk object failure, the predicted pose differs significantly by simple visual inspection.

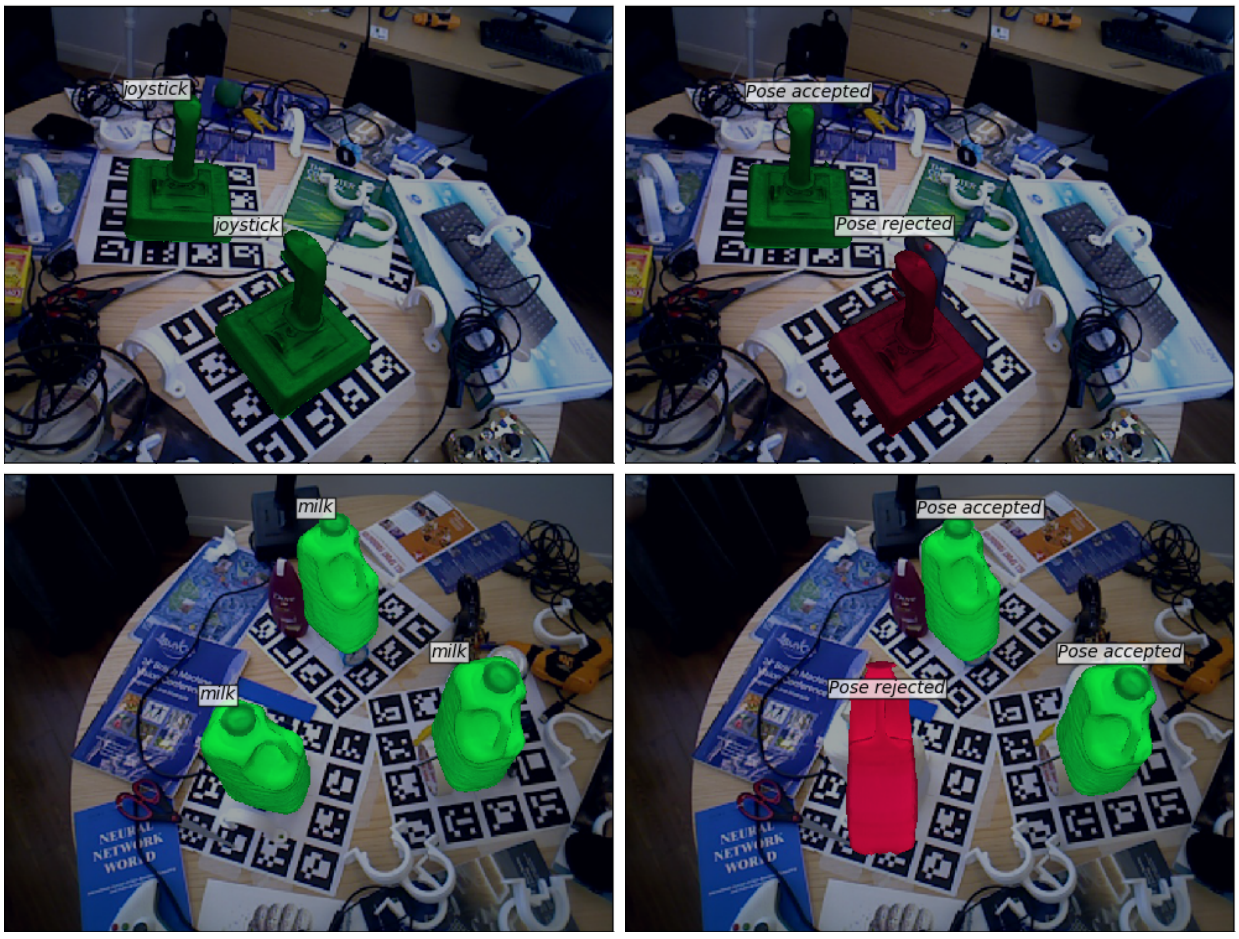


Figure 5.3: Example failure cases of the RGBD Network pose predictions under the **ADD** metric. The failure cases of the predicted poses (right) are projected in red. Again, the ground truth poses (left) are shown for reference.

5.1.3 Translation and Rotation Error

5cm5° Metric

Under this pose metric, the translation error and rotational error are evaluated in accordance with equations (2.6) and (2.9) respectively, and only poses with less than 5cm translation error and 5° of rotation error are accepted. The results are listed in table 5.3.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Average
RGB Network	13.1	3.2	23.3	20.7	23.2	20.0	17.3
RGBD Network	20.0	5.2	20.0	23.2	17.5	18.3	17.4
Deep-6DPose [5]	76.5	18.7	60.2	85.6	73.5	72.4	64.5

Table 5.3: 6D pose estimation test accuracy on the Tejani et al. dataset: **5cm5°**.

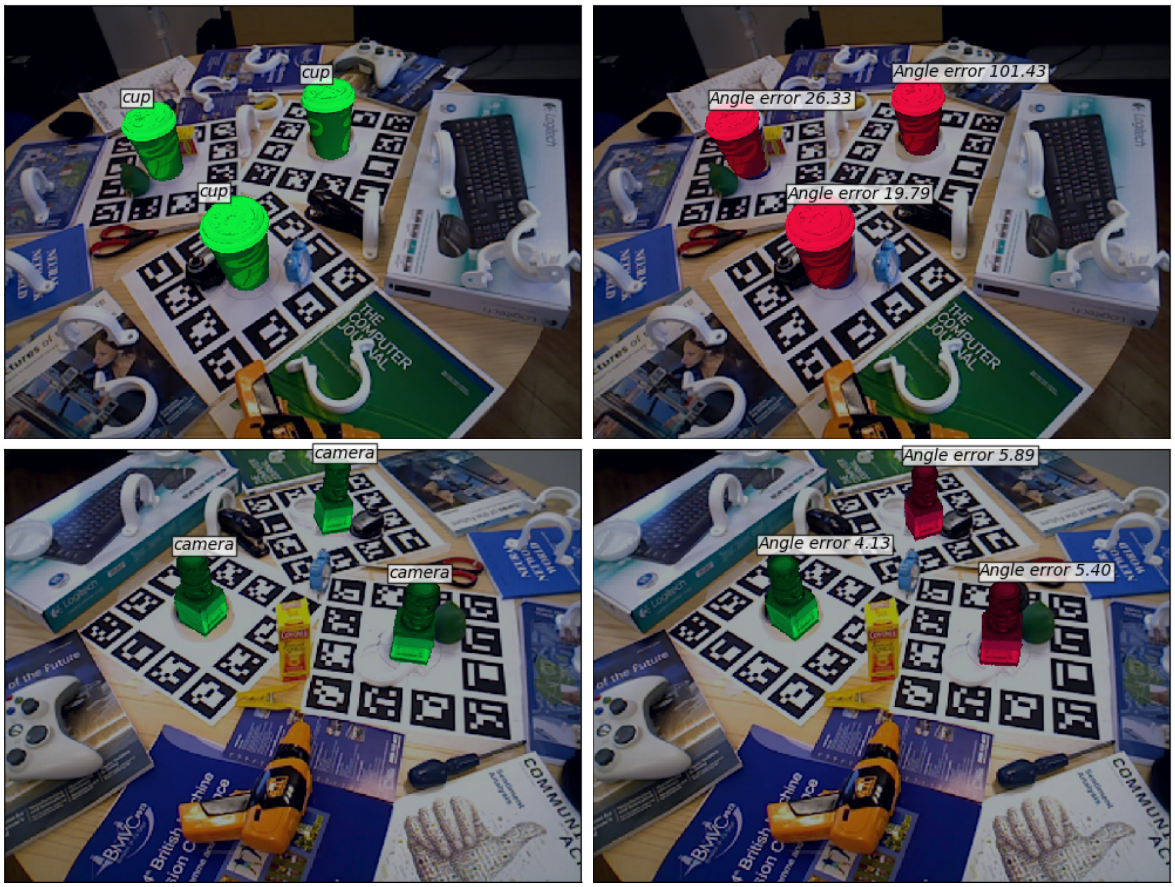


Figure 5.4: Failure cases on the **5cm5°** metric. Although the silhouettes of the estimated poses (right) align very well with the ground truth (left), the magnitude of the angle error (shown in the annotation) is too large in most of the pose estimates, causing them to be rejected. In some cases, it is challenging to distinguish the incorrect poses by the naked eye.

While the difference between the RGB and RGBD networks is insignificant under this metric, the performance compared to the state-of-the-art is significantly worse. Relaxing the angle threshold offers insight into both findings, revealing absolute rotational error as the major culprit.

Relaxed Angle Threshold

To gain additional insight into the performance of the networks, the 5cm5° metric was modified by relaxing the 5° angle threshold in steps of 5°, up to a maximum of 25°, and one additional measurement was performed without any angle threshold. The results are recorded in table 5.4 below.

	RGBD Network						
Angle Threshold	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Average
5°	20.0	5.2	20.0	23.2	17.5	18.3	17.4
10°	48.5	19.4	58.9	63.0	44.0	57.4	48.5
15°	55.0	35.4	80.3	80.5	56.5	73.1	63.5
20°	57.7	47.0	90.9	84.8	65.0	80.9	71.0
25°	58.6	57.0	93.4	86.5	69.0	85.3	75.0
None	65.8	76.6	96.8	92.4	82.0	93.3	84.5
	RGB Network						
5°	13.1	3.2	23.3	20.7	23.2	20.0	17.3
10°	35.7	18.2	62.8	57.4	49.0	52.3	45.9
15°	43.8	33.2	84.3	72.8	59.6	64.7	59.7
20°	45.2	45.1	91.7	76.3	64.1	72.7	65.9
25°	46.1	55.4	94.2	77.9	66.7	76.2	69.4
None	55.0	76.0	97.2	84.7	81.3	86.3	80.1

Table 5.4: 6D pose estimation test accuracy on the Tejani et al. dataset for varying angle thresholds. The translation error threshold is fixed at 5cm.

As the angle threshold gets relaxed, the margin between the depth-enabled and RGB-only network grows wider. These results suggest that although both networks perform similarly well on estimating highly precise poses, the RGBD network is more successful at pose estimates that meet less strict angle criteria, which is a plausible explanation for why the RGBD network has higher accuracy on the ADD metric. Note the case when the angle criterion is dropped completely, effectively turning the problem into 3D object detection - without an angle threshold,

all remaining error is due to a violation of the translation threshold, which turns out to be relatively insignificant compared to the rotational error. Interestingly, for both networks, the smallest object (camera) is also the most difficult to localize accurately.

5.1.4 2D Pose Metric

To evaluate the 2D pose metric, a 2D projection of a given object is performed using the predicted and ground truth pose. The IoU of the two resulting bounding boxes is computed, and the pose is accepted if the IoU is larger or equal to 0.5.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Average
RGB Network	96.4	97.6	91.7	94.7	96.5	92.0	94.8
RGBD Network	97.5	97.7	93.2	95.6	95.5	94.3	95.6
Deep-6DPose [5]	99.2	100	99.6	98.4	99.5	99.1	99.3

Table 5.5: 6D pose estimation test accuracy on the Tejani et al. dataset: **2D pose** metric.

As can be seen in table 5.5, both the RGB and RGBD networks achieve a relatively high accuracy of 94.8% and 95.6% respectively, although still lagging behind the state-of-the-art, which achieves virtually perfect accuracy on this metric. It is not surprising that both networks perform well on under the 2D pose criterion, as it is mostly sensitive to the translational component of the pose rather than the rotation - as evidenced by the results in the previous section, the majority of the pose error is due to rotation inaccuracy.

Due to this effect, the most dramatic discrepancy between any of the metrics can be observed in the *Coffee* sequence, where the RGBD network achieves the lowest accuracy under the 5cm5° metric yet the highest accuracy under the 2D pose criterion, further confirming that the network can accurately localize objects, yet it is much harder for it to infer accurate rotation angles. This can be likely attributed to the fact that the coffee cup exhibits the highest amount of symmetry out of all objects in the Tejani et al. dataset.

Figure 5.5: Example test cases evaluated on the **2D pose** metric. In the lower right example, a highly inaccurate predicted pose of the coffee cup is accepted by a narrow margin. The failure case in the upper image is rejected due to the fact that the shampoo is highly elongated along the *yaw* rotation axis and the predicted pose exhibits high rotation error around the *roll* axis.

As demonstrated in figure 5.5, the 2D Pose metric can be highly forgiving to inaccurate poses as long as the 2D object outlines align well enough. Therefore, it is important to understand the meaning behind the accuracy values, and the metric should be considered in contrast with other criteria rather than in isolation.

5.1.5 Visible Surface Discrepancy

Although VSD is perhaps the most resilient to pitfalls of pose evaluation, it is not often used in practice, possibly due to the number of parameters as well as implementation difficulty. Due to the lack of data, a comparison with state-of-the-art systems cannot be performed. Hence, table

5.6 only shows the VSD performance of the RGBD and RGB baseline networks. The Visible Surface Discrepancy was calculated in accordance with equations (2.10) and (2.11), and only poses with $VSD < 0.5$ were accepted as valid. The parameters δ and τ were set to the typical values of $15mm$ and $20mm$ respectively.

	Camera	Coffee	Joystick	Juice	Milk	Shampoo	Average
RGB Network	42.7	57.3	10.3	61.3	37.9	61.8	45.2
RGBD Network	51.9	58.0	9.3	69.9	35.0	68.9	48.9

Table 5.6: 6D pose estimation test accuracy on the Tejani et al. dataset: **VSD**.

The obtained results are mostly consistent with conclusions drawn from results of the other accuracy metrics. However, one notable difference is that the usually problematic *Coffee* sequence performs considerably better according to VSD, as this metric is robust to differences due to pose ambiguity. It can also be seen that the Joystick sequence achieves the lowest accuracy, consistently with the ADD metric, although the margin under VSD is much larger. However, it should be noted that VSD does not take relative object scale into account (since τ is kept constant), which penalizes larger objects. Since the joystick object has the second largest diameter of all at *235.6mm*, this very likely exacerbates the difference.

5.2 Network Speed

Table 5.7 shows the inference and training speed of the RGBD network as well as the RGB baseline. The depth-capable network has a slight speed disadvantage due to the overhead associated with the VGG-based depth feature extractor, which by itself is a relatively large network. Given that experiments were performed on a GPU which is 2 generations behind the state-of-the-art, it is plausible that newer models would enable the possibility of real-time 6D object pose estimation using these networks.

	Training	Inference
RGB Network	1.8 fps	6.0 fps
RGBD Network	1.4 fps	4.2 fps

Table 5.7: Training and inference speed on the NVIDIA Tesla M60.

	Inference Speed	GPU Model	GPU Architecture	CUDA Cores
RGB Network (Ours)	4.2 fps	Tesla M60	Maxwell (2014)	2048
RGBD Network (Ours)	6.0 fps	Tesla M60	Maxwell (2014)	2048
Deep-6DPose [5]	10.0 fps	TITAN X	Pascal (2016)	3584
BB8 [22]	3.4 fps	TITAN X	Pascal (2016)	3584
SSD-6D [12]	10.0 fps	GTX 1080	Pascal (2016)	2560

Table 5.8: Inference speed comparison.

In order to provide context, table 5.8 shows the inference speed of several notable 6D pose estimation systems in comparison to ours. When compared to the Tesla M60 [55], note that the Titan X [56] features 75% more CUDA cores while the GTX 1080 [57] has 25% more cores as well as a newer architecture, therefore the individual inference rates cannot be directly compared across the GPUs. However, it is clear that the performance of our networks is competitive with respect to leading 6D pose estimation systems.

In terms of the GPU memory requirements, the RGBD network uses 4.5 GB of RAM, whereas the RGB-only counterpart uses 4.0 GB. Again, the increased memory footprint is due to the necessity of storing a larger network as well as twice the amount of image data in GPU memory. However, from a practical point of view, the difference is insignificant thanks to the memory capacities of modern graphics processors.

5.3 Discussion

Based on the evaluation results on several standard pose error metrics, it can be concluded that the depth-capable network exhibits higher pose estimation accuracy than the RGB-only network. Although the difference between the two is not dramatic, it is almost certainly significant given the low variance in accuracy in the later stages of the training process. As revealed by analysis of the various pose error metrics, the difference in performance is primarily due to the RGBD network's higher rotation estimation accuracy. It should be noted that RGBD network utilizes 1.) a relatively straightforward way of merging knowledge from the color and depth channels and 2.) a depth feature extractor that was initialized with weights pre-trained

on color features. Improvements to either of these aspects would likely lead to further increases in accuracy.

The performance differences were not constant on all object sequences - in some cases, the depth-capable network demonstrated considerably higher accuracy; in others, the performance was comparable, with the RGB network taking a slight lead occasionally. Intuitively, the depth-capable network is expected to perform better when the depth channel captures pose information that is not present in the color channels. Whether that is the case can be contingent on shape and texture differences, lighting conditions as well as particular object poses. In the particular case of the Tejani et al. dataset, the 2 cases where depth did not help lead to an improvement was on the joystick and milk objects - both of which have the most irregular shapes which, despite lacking distinctive textures, carry rich edge information that conveys the poses accurately through the color channel. On the other hand, the RGBD network generally performs better on objects with more ambiguous poses, possibly by taking advantage of subtle cues present in the depth data. Consequently, the suitability of using the depth channel to increase pose estimation accuracy likely depends on the particular use-case, and prior knowledge about the nature of the objects of interest should be considered carefully.

Despite demonstrating the viability of using depth to increase object pose estimation accuracy, the RGBD network does not reach the performance of current state-of-the-art methods. Again, the differences are not consistent throughout the object sequences - instead, large performance differences were observed in performance on particular objects. We speculate that the performance discrepancy could be attributed to either of these two main factors (or their combination):

1. Shorter training time (about 10% of [5]). Although accuracy of the network seems to be saturating after the 50-epoch procedure, it cannot be ruled out that over a much longer time period, the performance on the problematic sequences converges to the accuracy achieved by [5].

2. The omitted semantic segmentation head that is present in Deep-6DPose [5] is indirectly responsible for the higher accuracy. Previous works have shown that the semantic seg-

mentation task aids the accuracy of 2D object detection [29] purely due to the additional term in the multi-task loss. It is plausible that a similar phenomenon could be observed in the context of 6D object pose estimation, as the segmentation loss term might aid the network in focusing its attention with pixel-level precision.

For further work, it would be of interest to investigate the reason behind the performance lag behind the state-of-the-art. However, given the project's resources, matching the performance of state-of-the-art is not one of the objectives of this work, which is primarily concerned with the comparison of the RGB and RGBD systems.

Chapter 6

Further Work and Conclusion

6.1 Further Work

Although this work has fulfilled its objective of assessing the objective of investigating the performance impact of including the depth image channel in an end-to-end 6D pose estimation network, many design choices taken throughout the project were affected by time and compute resource constraints. As a consequence, the noted improvements were deemed significant but not dramatic, and there are several avenues for potential further work that could lead to further performance improvements:

- *More advanced network architectures*: although there are numerous ways the RGBD network architecture could be changed, the most relevant ones relate to the aspects of depth feature extraction and integration - given enough training time and data, a depth feature extractor could be trained from scratch, likely resulting in more powerful features that are fully specialized for their input modality.

  Furthermore, different ways of integrating the depth and color channels could be investigated - currently, they are merged by a convolutional layer that follows directly after the feature extractors. This choice is quite arbitrary, and the merging process could take place e.g. after the RoI pooling layers, or just before the regression heads. Exploring the

effect of this design choice on the network's accuracy could yield useful insights into the problem.

- *Different training procedure:* In the current design, the entire network is trained using *joint approximate training* such that the entire network's weights are updated simultaneously for each backward pass by optimizing a single loss function. Although this is a simple approach that works well for training object detection systems, there is no guarantee that it is the optimal one for the problem at hand. Other, more sophisticated methods exist, wherein certain parts of the network are optimized while the rest of the network is frozen. This allows the optimization process to solely focus on one task, and the process can be repeated analogously with other parts of the network. Moreover, different loss functions and optimization methods are likely to have an impact on the convergence of the training process as well as the resulting accuracy.

- *Data augmentation:* One of the notable constraints in this work is the limited amount of training data, since the majority of the sequence images are kept aside for testing. One way to overcome this limitation is to augment the training data with synthetic images. This would entail rendering the object models in constrained random poses, and subsequently blending them with backgrounds. In previous works [58] it has been shown that this approach can be very successful, although great care must be taken in the generation and blending process in order to ensure robust knowledge transfer. Augmentation with synthetic data was considered in the initial project plan, but it was abandoned due to the significant time overhead of creating the image generation framework and training on additional data.

These options only outline some of the possibilities that could be explored in future work, and many other opportunities for improvement could be explored on the conceptual level, such as exploring different pose representations or novel pose refinement techniques.

6.2 Ethical, Legal, and Safety Concerns

No significant ethical, legal or safety issues pertaining to the project have been identified. The project is concerned with estimating poses of commonplace objects such as paper cups, as opposed to people or any controversial items. Hence, there is little potential for illicit, unethical or dangerous applications. Although one could argue that the resulting models could be adapted for more controversial domains, this would require significant effort invested into dataset creation and labeling, as well as retraining the system. Additionally, there is no unique aspect to this project that would make it more susceptible for adaptation for controversial uses compared to other related solutions.

6.3 Conclusion

In this final year project report, substantial background research was carried out on the topic of 6D object pose estimation, both in terms of existing work and associated challenges. Following on the background research, a deep neural network was implemented for the task of 6D object pose estimation from RGB images. This baseline architecture inspired by the Faster R-CNN object detection system was subsequently extended to utilize the depth channel of an image by direct transfer of knowledge from the domain of color feature extraction to depth feature extraction.

Both networks were trained and tested on the Tejani et al. dataset, which is one of the standard benchmarks for pose estimation. The testing accuracy has been evaluated on several pose metrics to obtain a holistic insight into the model's performance. Based on analysis of the results, it has been concluded that adding the depth channel improves network accuracy, particularly on objects that have more ambiguous poses, although the increase in performance is not dramatic. The network comes close to state-of-the-art accuracy for a few of the dataset's sequences, but lags behind in performance on the others. Several potential reasons are identified for the performance discrepancies that were observed, both with respect to the state-of-the-art as well as the RGB-only baseline. In terms of inference and training speed, the system is in line

with other comparable approaches. Further research directions are identified for closing the gap between the leading systems, as well as achieving better utilization of the depth channel.

Bibliography

[1] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[2] T. Hodaň, J. Matas, and Š. Obdržálek, "On evaluation of 6d object pose estimation," in *Computer Vision – ECCV 2016 Workshops* (G. Hua and H. Jégou, eds.), (Cham), pp. 606–619, Springer International Publishing, 2016.

[3] 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 *Computer Vision – ACCV 2012* (K. M. Lee, Y. Matsushita, J. M. Rehg, and Z. Hu, eds.), (Berlin, Heidelberg), pp. 548–562, Springer Berlin Heidelberg, 2013.

[4] T. Hodan, P. Haluza, S. Obdrzálek, J. Matas, M. I. A. Lourakis, and X. Zabulis, "T-LESS: an RGB-D dataset for 6d pose estimation of texture-less objects," *CoRR*, vol. abs/1701.05498, 2017.

[5] T. Do, M. Cai, T. Pham, and I. D. Reid, "Deep-6dpose: Recovering 6d object pose from a single RGB image," *CoRR*, vol. abs/1802.10367, 2018.

[6] S. Papert and M. I. of Technology. Artificial Intelligence Laboratory, *The Summer Vision Project*. AI memo, Massachusetts Institute of Technology, Project MAC, 1966.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural*

Information Processing Systems - Volume 1, NIPS'12, (USA), pp. 1097–1105, Curran Associates Inc., 2012.

[8] F. Rosenblatt, "The perceptron: A probabilistic model for information storage and organization in the brain," *Psychological Review*, pp. 65–386, 1958.

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

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

[11] Y. LeCun, B. E. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. E. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems 2* (D. S. Touretzky, ed.), pp. 396–404, Morgan-Kaufmann, 1990.

[12] W. Kehl, F. Manhardt, F. Tombari, S. Ilic, and N. Navab, "SSD-6D: making rgb-based 3d detection and 6d pose estimation great again," *CoRR*, vol. abs/1711.10006, 2017.

[13] P. J. Besl and N. D. McKay, "A method for registration of 3-d shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 239–256, Feb 1992.

[14] 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," *CoRR*, vol. abs/1607.06038, 2016.

[15] M. Martinez, A. Collet, and S. S. Srinivasa, "Moped: A scalable and low latency object recognition and pose estimation system," in *2010 IEEE International Conference on Robotics and Automation*, pp. 2043–2049, May 2010.

[16] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton, and C. Rother, "Learning 6d object pose estimation using 3d object coordinates," in *Computer Vision – ECCV 2014* (D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, eds.), (Cham), pp. 536–551, Springer International Publishing, 2014.

[17] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, pp. 381–395, June 1981.

[18] R. Kouskouridas, A. Tejani, A. Doumanoglou, D. Tang, and T. Kim, "Latent-class hough forests for 6 dof object pose estimation," *CoRR*, vol. abs/1602.01464, 2016.

[19] P. Wohlhart and V. Lepetit, "Learning descriptors for object recognition and 3d pose estimation," *CoRR*, vol. abs/1502.05908, 2015.

[20] O. H. Jafari, S. Karthik Mustikovela, K. Pertsch, E. Brachmann, and C. Rother, "The Best of Both Worlds: Learning Geometry-based 6D Object Pose Estimation," *ArXiv e-prints*, Dec. 2017.

[21] V. Balntas, A. Doumanoglou, C. Sahin, J. Sock, R. Kouskouridas, and T.-K. Kim, "Pose guided rgbd feature learning for 3d object pose estimation," in *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[22] 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," *CoRR*, vol. abs/1703.10896, 2017.

[23] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[24] J. R. R. Uijlings, K. E. A. van de Sande, T. Gevers, and A. W. M. Smeulders, "Selective search for object recognition," *International Journal of Computer Vision*, vol. 104, no. 2, pp. 154–171, 2013.

[25] C.-W. Hsu and C.-J. Lin, "A comparison of methods for multiclass support vector machines," *IEEE Transactions on Neural Networks*, vol. 13, pp. 415–425, Mar 2002.

[26] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[27] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013.

[28] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[29] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[30] G. Gkioxari, B. Hariharan, R. B. Girshick, and J. Malik, "R-cnns for pose estimation and action detection," *CoRR*, vol. abs/1406.5212, 2014.

[31] G. W. Hart, "Symmetry axes," 1996.

[32] F. Michel, A. Kirillov, E. Brachmann, A. Krull, S. Gumhold, B. Savchynskyy, and C. Rother, "Global hypothesis generation for 6d object pose estimation," *CoRR*, vol. abs/1612.02287, 2016.

[33] S. Hinterstoisser, V. Lepetit, N. Rajkumar, and K. Konolige, "Going further with point pair features," *CoRR*, vol. abs/1711.04061, 2017.

[34] T. Hoda, X. Zabulis, M. Lourakis, . Obdrlek, and J. Matas, "Detection and fine 3d pose estimation of texture-less objects in rgb-d images," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 4421–4428, Sept 2015.

[35] C. Altafini, "The de casteljau algorithm on se(3)," in *Nonlinear control in the Year 2000* (A. Isidori, F. Lamnabhi-Lagarrigue, and W. Respondek, eds.), (London), pp. 23–34, Springer London, 2001.

[36] D. Q. Huynh, "Metrics for 3d rotations: Comparison and analysis," *Journal of Mathematical Imaging and Vision*, vol. 35, pp. 155–164, Oct 2009.

[37] S. O. Tomas Hodan, Jiri Matas, "Measuring error of 6d object pose in the sixd challenge 2017." `https://github.com/thodan/sixd_toolkit/blob/master/doc/sixd_2017_measuring_error.pdf`, 2017.

[38] L. Torrey and J. W. Shavlik, "Transfer learning," 2009.

[39] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[40] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," in *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, (New York, NY, USA), pp. 675–678, ACM, 2014.

[41] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[42] F. Chollet *et al.*, "Keras." `https://keras.io`, 2015.

[43] Y. Chen, "A simple and fast implementation of faster r-cnn." `https://github.com/chenyuntc/simple-faster-rcnn-pytorch`, 2018.

[44] F. Research, "Visdom." `https://research.fb.com/downloads/visdom/`, March 2017.

[45] T. Hodan, "Sixd toolkit." `https://research.fb.com/downloads/visdom/`, December 2016.

[46] N. P. Rougier, "Glumpy," 08 2015.

[47] J. D. Hunter, "Matplotlib: A 2d graphics environment," *Computing In Science & Engineering*, vol. 9, no. 3, pp. 90–95, 2007.

[48] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," *CoRR*, vol. abs/1502.01852, 2015.

[49] R. Girschick, "Training r-cnns of various velocities." `https://www.dropbox.com/s/xtr4yd4i5e0vw8g/iccv15_tutorial_training_rbg.pdf?dl=0`, February 2015.

[50] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010* (Y. Lechevallier and G. Saporta, eds.), (Heidelberg), pp. 177–186, Physica-Verlag HD, 2010.

[51] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning," in *International conference on machine learning*, pp. 1139–1147, 2013.

[52] S. van der Walt, S. C. Colbert, and G. Varoquaux, "The numpy array: a structure for efficient numerical computation," *CoRR*, vol. abs/1102.1523, 2011.

[53] E. Brachmann, F. Michel, A. Krull, M. Y. Yang, S. Gumhold, and C. Rother, "Uncertainty-driven 6d pose estimation of objects and scenes from a single rgb image," in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 3364–3372, June 2016.

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

[55] "Next-Generation GPU-Powered EC2 Instances (G3) — AWS News Blog." `https://aws.amazon.com/blogs/aws/new-next-generation-gpu-powered-ec2-instances-g3/`.

[56] "NVIDIA TITAN X Graphics Card for VR Gaming — NVIDIA GeForce." `https://www.nvidia.com/en-us/geforce/products/10series/titan-x-pascal/`.

[57] "GeForce GTX 1080 Graphics Cards — NVIDIA GeForce." `https://www.nvidia.com/en-us/geforce/products/10series/geforce-gtx-1080/`.

[58] D. Dwibedi, I. Misra, and M. Hebert, "Cut, paste and learn: Surprisingly easy synthesis for instance detection," *CoRR*, vol. abs/1708.01642, 2017.