ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Learned Single Shot Image-Based Camera Localization

Master Thesis

Moyuan Zhou

September 15, 2019

Advisors: Dr. Bugra Tekin, Dr. Johannes L. Schönberger
Supervisor: Prof. Marc Pollefeys
Department of Computer Science, ETH Zürich

Abstract

great and wide applications in augmented reality, virtual reality, robotics, autonomous driving

We perform experiments on the synthetic SUNCG dataset and the recently released ScanNetv2 dataset

Acknowledgements

First and foremost, I would like to express my great gratitude to my advisors Dr. Bugra Tekin and Dr. Johannes L. Schönberger at Microsoft Research for their help this half year. They are very responsible and have provided countless advices to me during the whole period of my thesis. In this half year, we had weekly meetings regularly, and during each meeting they were responsive to the problems I was facing, gave me constructive suggestions and guided me to the potential directions with patience. The model we are using in this thesis also follows Bugra Tekin's work [9] last year.

I would like to show my special thanks to my supervisor Prof. Marc Pollefeys for offering the chance for me to work with Microsoft Research. I learnt quite a lot through this thesis and gained both theoretical knowledge and practical experience in camera localization and object pose detection area.

I am also particularly grateful for the great help given by my friend Zuoyue Li. When I have some theoretical or implementing problems related with deep learning, Leonhard cluster, etc, he can always offer me quite valuable suggestions and help me solve the problem.

Last but not least, I would like to thank my family especially my parents for their long-lasting spiritual and financial support throughout my studying life these years.

Contents

Chapter 1

Introduction

In recent years, image-based camera localization has been a key task in the areas of augmented reality, virtual reality, robotics, autonomous driving, etc. There are many methods relying on RGB-D cameras which are quite robust and accurate. However, the RGB-D cameras are power consuming, which makes approaches based on mobile and wearable cameras more attractive. In this thesis, we focus on camera localization from RGB images and aim for real-time, efficient, and robust camera localization.

Traditionally, the PnP [4] algorithm is used to solve the camera localization problem by computing the 6D camera pose given the 2D and 3D corresponding coordinates of multiple points. The fundamental problems of traditional approaches relying on local image features are textureless environments and robustness against strong changes in illumination/occlusion/viewpoint/etc. between the localized image and the given 3D model. To address these limitations, recently, deep learning has been applied to predict the 2D and 3D coordinates of these control points [1] and has achieved superior results. However, these approaches are typically very time-consuming to train and evaluate. In this thesis, we want to combine the benefits of both worlds and develop a learned approach that is efficient and robust.

The problem of 6D object pose estimation from RGB images is related to the problem of camera localization. Recently, an efficient, single shot approach [9] for simultaneously detecting an object in an RGB image and predicting its 6D pose without requiring multiple stages has been proposed. This approach resulted in an improvement over the state-of-the-art in terms of accuracy and efficiency, and addressed the challenges on keypoint occlusion and multiple object pose estimation. In this project, we aim to adapt this approach for efficient image-based camera localization. To this end, we will define keypoints on the 3D room layout for indoor environments and predict the projections of these 3D keypoints. Camera pose with respect to the object will then be computed using PnP [4] based on the correspondences

between 2D predictions and 3D reference keypoints. Camera pose with respect to the scene can also be computed given the location of the 3D object in the scene.

The major challenges of the project include limited data for training the localization task, occlusion, and motion. We have generated labels for the ScanNet and SunCG dataset for this work which can also be used by future work in related area. To address occlusion and motion issues, one initial idea is to increase the number of keypoints without slowing down the method, which is a direction to go for higher accuracy and robustness. [1] provided a trainable RANSAC approach for larger set of control points and can be integrated with the current model.

1.1 Motivation

In recent years, image-based camera localization has great and wide applications in multiple areas. Autonomous localization and navigation is necessary for a moving robot. Augment reality on images requires camera pose or localization. To view virtual environment, corresponding viewing angle needs to be computed.

Furthermore, unlike some other technics that require special devices, e.g. Lidar sensors, RGB-D cameras, etc, cameras are ubiquitous nowadays and people carry with their mobile phones that have cameras every day. Thus, we want to utilize only RGB images from 2D cameras to realize image-based camera localization.

As the current approaches are time-consuming [1] or can not be generalized to new scenes [10], we aim to come up with an efficient, real-time, and robust camera localization approach.

1.2 Background

1.2.1 Overview on localization tools

There are many localization tools nowadays, among which GPS is very commonly used outdoors, but it cannot be used indoors. Lidar, UWB, WiFi AP et al are effective indoor localization tools. However, they require special devices or data collections in advance. Compared to these tools, camera photos can provide higher discriminated features and more information, but in the same time require higher computation ability.

There are also many effective and robust methods relying on depth information acquired by RGB-D cameras currently. However, the RGB-D cameras are power consuming and not ubiquitous as 2D cameras. Thus, passive

RGB images that are more commonly used and easy to be acquired by mobile devices and wearable cameras become more attractive. In this work we focus on RGB images that could be captured by 2D cameras, and do not rely on depth information. Compared to other localization tools, image-based camera localization is the most flexible and low cost one.

1.2.2 Image-based camera localization

Image-based camera localization [11] is to compute camera poses under some world coordinate system from images or video captured by the cameras. Image-based camera localization can be classified into two categories according to that environments are prior or not: the one with known environments and the other one with unknown environments. Then one with known environments are usually the PnP problem studies, and the one with unknown environments consists of the methods with online and real-time environment mapping and the methods without online and real-time environment mapping. The former is commonly known as Simultaneous Localization and Mapping (SLAM) and the latter is the middle procedure of the commonly known structure from motion (SFM). In this thesis, we do not include any mapping or reconstruction procedure in our model since we are aiming for real-time localization given only single image as input, but for the training procedure 3D object model and object location in the scene is required as a prior knowledge.

There are also some approaches using convolutional neural network to predict camera pose directly from the 2D images or to compute 6D pose in some other way without using PnP algorithm. [10] predicts the orientation and translation of a camera given only a single picture. However, this approach is solving camera relocalization problem, and it can only predict in the same scene that the training period learnt. While this approach is useful in many robotic applications such as navigation and Simultaneous Localization and Mapping (SLAM), it cannot be generalised to a camera localization problem in a new/unseen scene. SSD-6D [3] relies on SSD architecture to predict 2D bounding boxes and a rough orientation estimate. Then based on the size of the 2D bounding box, it estimates the depth of the object and lift the 2D detection to 6D. However, the refinement step of this approach increases the running time a lot that cannot make a real-time prediction feasible.

Recently, a real-time single shot 6D object pose estimation approach [9] has been proposed. 6D object pose estimation is related with camera pose estimation which let us see a possibility of realizing real-time camera localization task in a general scene. [9] is using a single shot deep convolutional neural network to predict the 2D projections of the object's 3D bounding boxes, and then use a PnP algorithm to compute 6D object pose given the corresponding 2D coordinates and locations of 3D points. We adapt this

approach for our camera localization task. Specifically, we use a PnP algorithm to calculate camera pose with respect to the object from the 2D coordinates predicted by the network and the corresponding 3D keypoints of some known 3D models. Usually, the ground truth of the

1.2.3 PnP algorithm

As we are applying PnP algorithm to calculate the camera pose according to some corresponding 2D and 3D points. Here is a introduction to PnP algorithm.

Camera pose determinations from known 3D space points are called perspective-n-point problem, namely PnP problem. Let n be the number of used points. When $n \geq 6$, the problem is linear. When $n = 3, 4, 5$, the problem is nonlinear. And when $n < 3$, there is no solution. Although the P3P problem has been well solved, but there may exist multiple solutions. In this work, we set the number of keypoints per object as 9. Although there may sometimes be outliers due to occlusion or other reasons, we can still guarantee it is a linear problem at most of the cases.

When the PnP problem is linear, there are also a lot of works studying on efficient optimizations for the camera poses from small number of points. [4] provide an accurate O(n) solution to the PnP problem, called EPnP which is widely used today. In our approach, we also utilize EPnP and compare it with other PnP solutions.

1.2.4 6D object pose estimation

There are also lots of current works on 6D object pose estimation. Traditional approaches are mainly local image features extraction and matching. There are many fast keypoint and edge-based methods for textured objects, but not effective to weakly textured or untextured objects, or low resolution video streams.

Recently, deep learning methods are utilized in solving 6D object pose estimation problem and have achieved outstanding performance. BB8 [6] is a 6D object detection pipeline made of one CNN to coarsely segment the object and another to predict the 2D locations of the projections of the object's 3D bounding box given the segmentation. It then use PnP algorithm to compute 6D pose using the corresponding points. The method is effective but slow due to its multi-stage nature. SSD-6D [3] is another approach which relies on SSD architecture to predict 2D bounding boxes and a very rough estimate of the object's orientation in a single step. Then the object's depth is approximated from the size of the 2D bounding box in the image and the 2D detection is lifted to 6D. However, both BB8 and SSD-6D methods require

a post-processing step to refine the result, which increase the running time linearly and make the methods not feasible for real-time tasks.

1.2.5 Constraints on image-based camera localization problem

The common problem for camera localization or object pose detection is that the datasets available are rather limited, compared to datasets for other tasks like classification or tagging. Specifically, the most common datasets for detection contain thousands to hundreds of thousands of images with dozens to hundreds of tags, while classification datasets have millions of images with tens or hundreds of thousands of categories. Labeling images for detection is far more expensive than labeling for classification or tagging where tags are often user-supplied for free. These result in object detection problem lack of enough datasets and labels.

In this work, we generate the 2D projections of object's 3D bounding boxes for multiple objects as labels for indoor environments. We take ScanNet dataset as the realistic dataset and SunCG together with six tools for synthetic dataset. The labels generated can also be used in future works on related topics.

1.3 Focus of this work

As clarified in the previous background section, we are aiming for a real-time and robust image-based camera localization approach that can predict the camera pose in a new scene. The new scene could be in similar environment as the training scenes, e.g. indoor environment with furnitures, but they should not be the same scenes. The approach should also be able to generalise to various object with no need of precise and detailed texture.

We adapt the recent real-time single shot 6D object pose prediction approach [9] to our camera localization approach. We also generate new labels for indoor environment dataset. Basically, the approach uses a single shot deep CNN architecture to find the corresponding 2D image coordinates of the 3D ground keypoints for some known 3D models in the environments and then applies a PnP algorithm to calculate the camera pose with respect to the object. We choose the 3D keypoints as the 8 corners of the object's 3D bounding box plus the center point of the object. The network is single shot and end-to-end trainable, and the model is also accurate without requiring any post-processing, so it is faster than the other current existing methods which are multi-stage or require post-processing for pose refinement.

Chapter 2

Related Work

In this chapter we review the existing works on camera localization from classical feature and template matching methods to newer end-to-end trainable CNN-based methods. Since we use 6D object pose estimation to realize our camera localization, object detection and 6D pose estimation related works are also reviewed.

2.1 Camera localization methods

2.1.1 Classical methods

Traditional RGB object instance recognition and pose estimation works are mainly local feature extraction and mapping.

2.1.2 CNN-based methods

There are some CNN-based end-to-end camera pose prediction methods.

2.2 Object detection based methods

Camera location can also be computed from 6D object pose. 6D object pose detection approaches are mostly based on object detection means. Here we review related works from the initial R-CNN based object detection methods to the latest single shot 6D object pose detection method.

2.2.1 Object detection

R-CNN and further methods propose various locations and regions

SSD

YOLO YOLO is a rael-time single shot object detection approach and is proposed in the same time as SSD. Most of the works for object detection before YOLO have a first step to propose possible regions of the object, and then do the classification and refinement. The multiple stages make the model hard to optimize since each component must be trained separately. The post-processing step also makes the training and prediction rather slow.

YOLO frame object detection as a regression problem instead of the previous classification problem. Instead of doing classification on a proposed region, YOLO can predict the location of the bounding boxes of the object directly. Given a full image, YOLO use a single network to predict all bounding boxes across all classes for the image simultaneously in one evaluation. Thus, YOLO realizes a single shot, end-to-end trainable network for object detection task.

Although YOLO cannot achieve the state-of-the-art accuracy but it is extremely fast and capable for real-time object detection. Compared with other real-time systems, it has more than twice of the mean precision. Although YOLO has lower recall compared to region proposal-based methods, it has less than half false positives on background errors compared to Fast R-CNN.

There are some constraints of YOLO as well. The first one is that there is a limit on the number of nearby objects that the model can predict. Since YOLO partition the image into multiple grid, and each grid cell can only predict one class, although it can predict B bounding boxes per grid, there is still only one object predicted as output. If two objects are very close and have their center in the same grid, only one of them can be predicted. The second one is that YOLO does not achieve the state-of-art accuracy. There is significant number of localization errors and relatively low recall of YOLO compared to region proposal-based methods.

YOLO v2 YOLO v2 has done some improvements to the YOLO detection method. Inspired by Faster R-CNN, which use the region proposal network(RPN) to predict offsets and confidences for anchor boxes, YOLO v2 removed the fully connected layer at the end of the network, and instead use anchor boxes to predict bounding boxes. YOLO v2 decouples the class prediction mechanism from the spatial location and instead predict class for every anchor box. In this way, the limitation of only one class is predicted per grid cell is released, and now we can better predict nearby objects.

YOLO v2 achieves a good tradeoff between speed and accuracy. It outperforms Faster R-CNN with ResNet and SSD while still running significantly faster. In this work, our network architecture is also based on YOLO v2.

2.2.2 6D object pose detection

BB8 BB8 [6] is a 6D object detection approach which consists of one CNN to realize a coarse segmentation and another to predict the 2D locations of the projections of the object's 3D bounding box, which are then used by a PnP algorithm to compute the 6D object pose. The segmentation stage generates a 2D segmentation mask for presenting a cropped image to the second network. There is also an optional additional step that refines the predicted poses. The method is slow due to its multi-stage nature.

SSD-6D The SSD-6D [3] approach relies on the SSD architecture to predict object's 2D bounding boxes and a pool of the most likely 6D poses for that instance. It then predicts the approximated depth of the object from the size of the 2D bounding box in the image and lift the output from 2D to 6D object pose. In the final step, the approach refine each pose in every pool and select the best after verification. As the method require a refinement step to get a good accuracy, the running time is increased linearly with the number of objects being detected.

Real-time single shot approach The recently proposed real-time seamless single shot approach [9] performs 6D object pose prediction in an RGB image without multiple stages or hypotheses. It proposed a new CNN architecture which is inspired by YOLO and YOLO v2. The end-to-end trainable architecture makes it easy and fast to optimize. The approach turns out to be fast enough for real-time 6D object pose detection and accurate without requiring any additional post-processing.

The network is based on the YOLO v2 network but extends 2D detection to 6D detection task. It directly predicts the 2D image locations of the projected vertices of the object's 3D bounding box and then use a PnP algorithm to predict the object's 6D pose.

Compared with other approaches, the single shot approach outperforms BB8 [6] and SSD-6D [3] when they are tested without post-processing - as we introduced above, BB8 and SSD-6D have pose refinement step as post-processing to boost the accuracy but with a cost of much slower performance. When handling multiple objects, the single shot approach virtually has no time-penalty, that is the running time remains constant, whereas other methods grow proportional to the number of objects.

Chapter 3

Methodology

This work is following B. Tekin's real-time single shot 6D object pose detection method [9] which is inspired by the performance of YOLO [7] and YOLO v2 on single shot 2D object detection. The network architecture is based on YOLO v2 but extends 2D detection to 6D detection task. The output of the network is the 2D coordinates of the projections of the 3D keypoints of the object which is then applied to a PnP algorithm to compute the camera pose together with the ground 3D points.

In this section, we first review the network architecture of some single shot object detection methods, we take YOLO and YOLO v2 for examples, and then elaborate our improvements on it for a camera localization problem.

3.1 Single shot 2D object detection

The network architecture of our work is based on YOLO v2 but is amenable to other single shot detectors such as SSD and its variants. In this section we briefly introduce the network architecture of YOLO and YOLO v2.

YOLO [7] is a single shot object detection approach which first frames object detection as a regression problem to spatially separated bounding boxes and associated class probabilities, instead of the prior repurposing to classification works.

YOLO uses a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation. Given an input image, the system divides the image into an $S \times S$ grid. The grid that the center of an object falls into is responsible for detecting that object. Each grid cell predicts B bounding boxes and confidence scores for those boxes. The confidence score reflects how confident the model is that the box contains an object and also how accurate it thinks the box that it predicts is. YOLO takes the intersection over union (IOU) as the measurements for the confi-

dence score. Formally, the confidence is defined as $Pr(Object) * IOU_{pred}^{truth}$. That is, if no object exists in the cell, the confidence is 0. Otherwise the confidence score equals the intersection over union (IOU) between the predicted box and the ground truth. Each grid cell also predicts C conditional class probabilities, $Pr(Class_i|Object)$. These probabilities are conditioned on the grid cell containing an object. There is only one set of class probabilities predicted per grid cell, regardless the number of bounding boxes B is predicted per cell. Thus, only one object is detected per grid cell. This limits YOLO from detecting nearby objects that have their centers in the same cell.

Each bounding box consists of 5 predictions, where 4 of them represents the coordinates of the center of the box and the width and height of the box, and plus a confidence prediction. Thus, the output of the network is a $S \times S \times (B * 5 + C)$ tensor, where B is the number of bounding boxes predicted per grid cell and C is number of classes.

Network

YOLO consists of 24 convolutional layers to extract features from the image and 2 fully connected layers to predict the output class probabilities and bounding box coordinates. it also designed a pretrain network which consists 20 convolutional layers, an average-pooling layer and a fully connected layer for pretraining the network.

Training

At training time only one bounding box predictor is wanted to be responsible for each object, so one predictor is chosen based on which prediction has the highest current IOU with the ground truth. YOLO uses the sum squared error as its loss function with some modifications. Firstly, there should be a weight to distinguish localization error with classification error. Thus, YOLO increases the loss from bounding box coordinate predictions. Secondly, since there are many grid cells not containing any object, which pushes the confidence score towards zero, the loss from confidence predictions for boxes that don't contain any object is decreased. Thirdly, deviations in large boxes should also matter less than in small boxes, so the square root of the bounding box width and height is predicted instead of width and height directly. Finally, the loss function only penalizes classification error if an object is present in that grid cell and only penalizes the bounding box coordinates error if an object is present in that grid cell and if that bounding box has the highest IOU among other predicted bounding boxes in the grid

cell. Specifically, the loss function is defined as:

$$\begin{aligned}
&\lambda_{cood} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] \\
&\quad + \lambda_{cood} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} \left[(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2 \right] \\
&\quad + \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{i,j}^{noobj} (C_i - \hat{C}_i)^2 \\
&\quad + \sum_{i=0}^{S^2} \mathbb{1}_{i}^{obj} \sum_{c \in class} (p_i(c) - \hat{p}_i(c))^2
\end{aligned} \tag{3.1}$$

where $\mathbb{1}_i^{obj}$ denotes if object appears in cell i and $\mathbb{1}_{i,j}^{obj}$ denotes that the jth bounding box predictor in cell i has the highest IOU among other predicted bounding boxes in the grid cell. λ_{coord} is set to 5 and λ_{noobj} is set to 0.5.

To avoid overfitting, YOLO introduce a dropout layer with rate 0.5 after the first connected layer. Data augmentation is also applied by random scaling and translations of up to 20% of the original image size.

Inference

At test time, YOLO computes the class-specific confidence for each bounding box by multiplying the conditional class probabilities with the individual box confidence predictions. Same as training, predicting detections requires only a single network evaluation, which turns out to be much faster than those classifier-based methods. Sometimes there are large objects or objects near the border well predicted by multiple grid cells, non-maximal suppression is used when an object is localized by multiple cells.

YOLO v2

YOLO v2 has done some improvements to YOLO based on some ideas from other high accuracy object detection methods such as Faster R-CNN []. It adds batch normalization to replace the dropout layer and achieves a better convergence. It also increases the resolution of input images for training and fine tunes the network to adjust its filters to work better on higher resolution input. Finally, it removes the fully connected layers from YOLO and uses anchor boxes to predict offsets instead of the coordinates for the bounding boxes. In the same time, the class prediction mechanism is also decoupled from the spatial location so that the network can predict class for every anchor box and realize multiple objects prediction per grid cell.

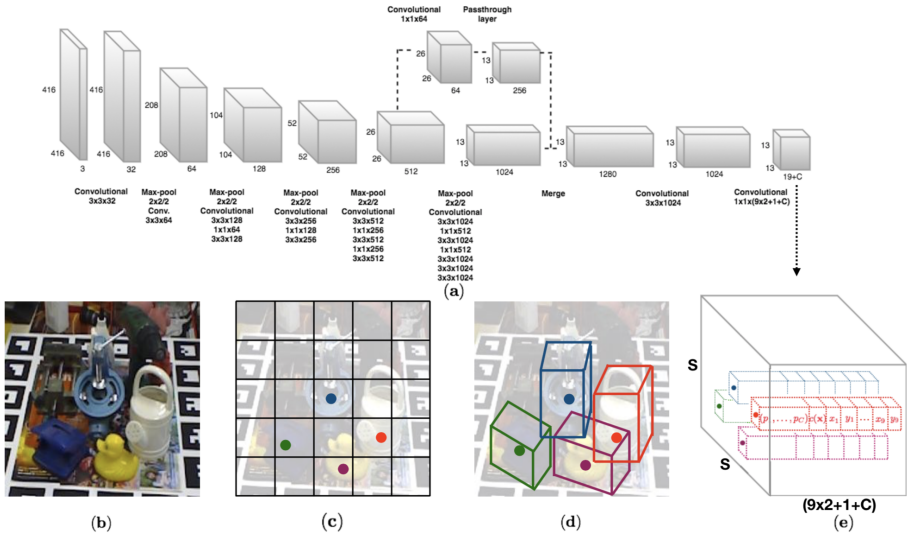


Figure 31: Network architecture from [9]: (a) The proposed CNN architecture. (b) An example input image with four objects. (c) The $S \times S$ grid showing cells responsible for detecting the four objects. (d) Each cell predicts 2D locations of the corners of the projected 3D bounding boxes in the image. (e) The 3D output tensor from our network, which represents for each cell a vector consisting of the 2D corner locations, the class probabilities and a confidence value associated with the prediction.

3.2 Model

Our model follows B. Tekin's model [9] which performs 6D object pose detection based on YOLO v2 architecture. Since YOLO's network is designed to regress only 2D bounding boxes, a few more 2D points had to be added for 6D object detection. In our model, a CNN-based network predicts 2D projections of the corners of the 3D bounding box around the object. Then a PnP algorithm is used to compute camera pose with respect to the object efficiently given the 2D coordinates and 3D ground control points, i.e. the object's 3D bounding box corners.

We choose the keypoints of the 3D object model as the 8 corners of the object's tight 3D bounding box plus the centroid of the object's 3D model. Thus, we parameterize the 3D model of each object with 9 control points. The control points are guaranteed to be well spread out in the 2D image. In practice, the control points can be defined in other ways as well.

Given a single full color image as input, our model processes it with a fully convolutional architecture shown in Figure 31 and divides the image into a 2D regular grid containing $S \times S$ cells same as YOLO does. Each grid in our model is associated with a multidimensional vector, which consists of the

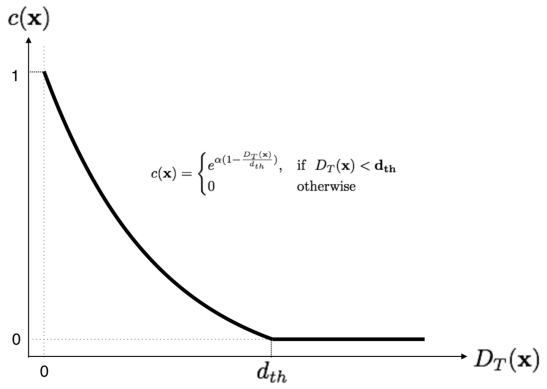


Figure 32: Confidence function

predicted locations of the control points, class probabilities of the object and an overall confidence score. When N objects are present in different cells, we have N such vectors. we train our network to predict these target values. Thus, the output of our network is a 3D sensor of size $S \times S \times D$, where D is the size of the multidimensional vector. In our case, $D = 9 \times 2 + 1 + C$, where the location of each 2D control point consists 2 coordinates and C is the number of classes our model can predict. Same as YOLO, the conditional class probabilities is conditioned on the cell containing an object.

In YOLO the confidence value is calculated by the intersection over union (IOU) between the predicted box and the ground truth. However, we have 3D objects in our case and it is not easy or fast to calculate the IOU over 3D voxel cuboids. In order not to slow down the training procedure, we take a confidence function to model the confidence value as introduced in [9]. As shown in Figure 32, the confidence function c(x) returns a confidence value for a predicted 2D point, denoted by x, based on its distance $D_T(x)$ from the ground truth 2D point. Formally, the confidence function is defined as follows:

$$c(x) = \begin{cases} e^{\alpha(1-\frac{D_T(x)}{d_{th}})}, & \text{if } D_T(x) < d_{th} \\ 0 & \text{otherwise} \end{cases} \tag{3.2}$$

where $D_T(x)$ is the 2D Euclidean distance in the image space, d_{th} is a cut-off value d_{th} so that if the distance is no less than it, simply set the confidence score to 0, and the parameter α defines the sharpness of the exponential function. The overall confidence score of the predicted bounding box is then computed as the mean value of the confidence value of all the control points.

Our network is based on YOLO v2's network and consists of 23 convolutional layers and 5 max-pooling layers. Similar to YOLO v2, we choose

$S = 13$. As the network downsamples the image by a factor of 32, the input resolution is set to 416×416. We also added a passthrough layer to let the higher layers of our network use fine-grained features.

Our model also supports detecting multiple objects in one grid cell. When multiple objects are close to each other, their centroid may fall in the same grid cell. We allow each grid to predict up to 5 bounding boxes so that multiple nearby objects or occluded objects can be detected. As in YOLO v2, we precompute five anchor boxes that define the size of a 2D rectangle tightly fitted to a masked region around the object in the image. During training, we assign whichever anchor box has the most similar size to the current object as the responsible one to predict the 2D coordinates for that object.

3.3 Training

During training, the confidence value is computed on the fly using the confidence function in Eq. 3.2 to measure the distance between the current coordinate predictions and the ground truth. Each grid cell predicts the offsets for the 2D coordinates with respect to (cx, cy), the top-left corner of the cell. We constrain the centroid within the associated grid cell, i.e. the offset of the centroid to lie between 0 and 1. We do not constrain the network's output for the corner points as those points should be allowed to fall outside the cell.

Our loss function is the sum of the loss from all of the three components - coordinate loss, classification loss, and loss from confidence value. We use mean squared error for the coordinate and confidence value losses, and use cross entropy for the classification loss. In this way, we don't need to increase the weight for coordinate loss any more to distinguish coordinate loss with classification loss as suggested by YOLO. Similar to YOLO, we downweight the loss from confidence value for cells that don't contain objects, and increase the weight of confidence loss for cells that contain objects. This improves the model's stability. Formally, the loss function is defined as:

$$L = \lambda_{pt} L_{pt} + \lambda_{coof} L_{coof} + \lambda_{id} L_{id} \tag{3.3}$$

where L_{pt}, L_{coof}, L_{id} denote the coordinate, confidence and the classification loss respectively. λ_{coof} is set to 0.1 for cells that don't contain objects and to 5 for cells that contain objects. λ_{pt} and λ_{id} are set simply to 1.

3.4 Prediction

At test time, the prediction procedure invokes the network only once. We compute the class-specific confidence scores for each object by multiplying

the conditioned class probability and the confidence value of the prediction returned by the confidence function, and cells with predictions with low confidence values are pruned. For large objects or objects whose centroid lies close to the border of grid cells, multiple grid cells may have good prediction of it. In order to achieve a more robust and accurate model, we inspect all gird cells in a 3×3 neighborhood of the cell having highest confidence score. We compute the weighted average of the predictions of the 2D locations according to the confidence scores.

At run-time, the network outputs the 2D coordinates prediction of the projections of the object's control points together with the classification. We can estimate the 6D camera pose with respect to the object using a PnP algorithm with the corresponding 2D and 3D points. After we get the 6D camera pose with respect to the object, we can compare the camera pose with respect to the scene by applying the transform of the object's position in the scene.

3.5 Implementation Details

We train our network on Leonhard cluster. Since we have limited memory assigned to each GPU node on the cluster, we can only train with batch size of 8. We run the training for 700 epochs and select the best weights according to the accuracy in validation data. For optimization, we have experimented both SGD and Adam algorithms. It turns out Adam results in a better and more robust prediction. We set the learning rate to 0.001. For the PnP algorithm, we tested on both with RANSAC and without. It turns out the result does not differ a lot, indicating that our predictions for the 9 control points seldom contain outliers. For the error metrics, in addition to rotation and translation errors of camera pose, 2D euclidean error for all of the projected vertices and control points, we also have accuracy of 2D and 3D vertices that have an error prediction with a threshold and we calculate the error of corner indices ordering as well.

Chapter 4

Datasets

Dummy text.

4.1 Dataset types

In this section, we introduce the datasets we have used or have considered to use in our work. We divide the datasets in two parts, synthetic and real. Our idea is to use synthetic data to help with training and realize a better prediction on real data.

4.1.1 Synthetic data

For synthetic data we have used SunCG dataset and SIXD toolkit to render some simple views of single object from SunCG .

SIXD Toolkit

SIXD toolkit is implemented by Hodan et al to facilitate participation in the SIXD Challenge, of which the goal is to evaluate methods for 6D object pose estimation of a rigid object from RGB or RGB-D images and to establish the state of the art. The challenge results are included in their work BOP: Benchmark for 6D Object Pose Estimation [**?**].

The toolkit provides functions of rendering a given object in different camera views and supports user customized textures. Thus, we use it to render object in different camera views and randomize the texture of the object and generate some synthetic images for training and testing in our work.

However, section 6.2.1 on page 37.

Figure 41: SIXD rendering

Figure 42: SIXD rendering of sofa in random texture with random background

SunCG

SunCG dataset is a manually created large-scale dataset of synthetic 3D scenes with dense volumetric annotations [**?**]

4.1.2 Real data

We aim to have a nice generalization of our model to real data. Thus, we mainly focus on the prediction result of our model on datasets consisting real data. In this subsection, we introduce some current datasets containing real data of indoor environments, and elaborate whether they are capable to be used in our work.

NYU-Depth

The NYU-Depth V2 dataset is introduced by Silberman et al and originally generated for image segmentation [5]. It is comprised of video sequences from a variety of indoor scenes as recorded by both the RGB and Depth cameras from the Microsoft Kinect. It features 1449 densely labeled pairs of aligned RGB and depth images among 464 scenes taken from 3 cities plus 407,024 unlabeled frames. Each object in the data is labeled with a class and an instance number.

However, the dataset does not provide camera pose information or camera extrinsic parameters for each image, so there is no ground truth for camera location. It also doesn't provide any reconstructed 3D scenes or models, so that we cannot generate ground truth of the 2D coordinates labels for the corresponding 3D keypoints. Thus, NYU-Depth dataset is not applicable for our training or testing procedure.

7-Scenes

The 7-Scenes dataset provided by Microsoft Research is a collection of tracked RGB-D camera frames recorded from a handheld Kinect at 640×480 resolution. It uses KinectFusion system to obtain the ground truth camera tracks and a dense 3D model. Many works and applications in relocalization [8, 2, 10] and dense tracking and mapping use 7-scenes for evaluation. While 7-Scenes are available for end-to-end camera pose learning methods, it does not suit for our object pose detection based method.

For the training procedure and labeling, our method needs the object's 3D model and position with respect to the camera as a prior knowledge. 7-Scenes has provided the dense reconstruction represented by signed distance volume of each scene. However, there is no category or object instance labeling related with the 3D scenes. Thus, we cannot get the 3D model and corresponding 2D labels for each object and train for 6D object pose

Figure 43: SunCG of sofa in random texture with random background

detection. Also, there are limited number of scenes, 7 exactly, which is not enough for our training procedure, where we aim to generalize to as many as possible different scenes.

During testing, we do not need the 3D object model any more for 6D object pose detection as long as there is trained category object exists in the image. However, our estimation of camera location is with respect to the object. If we have knowledge of the object 3D position with respect to the scene, we can also calculate the camera pose with respect to the scene. 7-Scenes only provides ground truth camera pose with respect to the scene, but do not have any knowledge of the object position. Thus, we cannot compute the camera location with respect to the scenes and 7-Scenes dataset is not applicable for evaluation in our method.

ScanNet

ScanNet is originally introduced by Dai et al in 2017 for 3D reconstructions of indoor scenes [**?**]. ScanNet is an RGB-D video dataset containing 2.5 million views in more than 1500 scans, annotated with 3D camera poses, surface reconstructions, and instance-level semantic segmentations. To collect the data, an easy-to-use and scalable RGB-D capture system includes automated surface reconstruction and crowdsourced semantic annotation is designed. Using this data helps achieve state-of-the-art performance on several 3D scene understanding tasks, including 3D object classification, semantic voxel labeling, and CAD model retrieval.

The instance-level semantic segmentation on the 3D reconstructions helps us to generate our labels in this work. We can use the segmentation information to extract each object and find the tight 3D bounding box of it. Since the object is already in its location with respect to the scene, we can directly use the camera pose provided correspondence to each RGB-image to compute the 2D projections of the 9 control points of the 3D model. However, if we want to compute the camera pose with respect to the scene, we need to save the object position in the scene, and apply the reverse of it after we get the camera pose with respect to the object from PnP algorithm. During evaluation, we would need a 3D model of the object or the locations of the bounding box of the 3D model to compute the camera pose with respect to the object using PnP algorithm. If the object position with respect to the scene is given or saved, we can also compute the camera extrinsic parameters and compare with the ground truth camera pose. Thus, ScanNet is good for both training and testing procedure in our work, and we can use it alone for our training procedure or mix it with SUNCG or sixd rendering images. We use it as our main testing dataset and aim to realize a good camera localization on this real data of indoor environments.

limitations of ScanNet However, during implementation there are some limitations due to the drawbacks of ScanNet dataset. The error of our labeling is mainly due to the incomplete reconstruction of the 3D models.

Firstly, the bounding box of the 3D model is incorrect due to partly reconstruction of some objects. Some of the bounding boxes may turn out to pass through the object instead of bound it. As shown in figure ?, the legs of the chair is not fully constructed in its 3D mesh, and the result labeling shows an error bounding box. This incomplete reconstruction usually happens to the thin or tiny part of the object, due to the noises from scanning. Also, the incompleteness may frequently occur to the back and downside of the object or the side of the object against the wall or facing the floor. Those sides are not well scanned due to hard access of the camera, so they are often poorly reconstructed.

Secondly, the bounding box is not well aligned with the axis. Since this dataset is from real data, there is no existing object model as the synthetic dataset provides. We extract the 3D model of the object from the 3D reconstruction of the whole scene using the instance-level segmentation information provided by ScanNet. Thus, our extracted 3D model may not be aligned with the axis. Since our 3D bounding box is calculated by the minimum and maximum values of each axis, we need to rotate the 3D object model to be aligned with the axis to get the tightest bounding box. Since all of the reconstructed object has a upright direction of z = 1, this makes our job easier. We only need to find out the direction in xy plane for aligning the object. In the ideal case, our initial idea is to use PCA to get the axis directions. However, due to the error and noises introduced by reconstruction, the object may not be symmetrically reconstructed. Sometimes one leg of the four legs of the table is missing, leading the PCA direction to be the diagonal of the top of table rectangle. Even though the object is completely reconstructed, the number of points are not equally distributed on the object mesh - one side may have mesh denser vertices than the other side. This also leads to the PCA not aligned with the ideal direction. To address this problem, we rotate the model in xy plane from -30 degree to 30 degree and calculate the new bounding box per 5 degree gap and take the smallest bounding box we ever get. This solves the problem to a large extent, but not very accurate since we set the granularity to 5 degree in trade off with the processing speed.

Thirdly, some reconstruction has a distortion. As we can see from figure , the floor of the whole scene is not flat. Thus, the object model is not accurately aligned with z axis sometimes. We can also do the same trick to rotate the object in zx and yz plane, and get the tightest bounding box, but the result is not always better. The main reason is due to incomplete reconstruction as well. As we can see in figure , the most frequently incomplete reconstruction occurs at the bottom back side of the object. When we rotate the couch

in yz plane (we assume the longest side is aligned to x axis), the tightest bounding box becomes the one passing though the actual object but bound to the incomplete mesh. Thus, in either way, whether we refine the upright direction or not, we cannot get an ideal result due to the noises introduced by the scanning and reconstruction.

Lastly, there is some noise on camera calibration and camera pose. Camera calibration is not very stable for all the scans - each scan may have a bit deviation on camera intrinsic parameters. However, in training and testing time we do not change camera intrinsic parameter depending on the scan that the input image belongs to. Therefore, some noises may be generated from this small error.

4.1.3 Dataset processing

In this subsection, we elaborate some aspects we need to note during generating labels and filtering out training and testing data.

Order of corners Since our network is to predict the 2D projections of the object's 3D control points, the order of the points is important to be consistent in order not to confuse the network. When we mix the training dataset from ScanNet and SunCG, consistent order of control points between different dataset on the same class object is also necessary. Since in ScanNet the object model is not provided, but we have to extract it from the 3D scene reconstruction according to instance segmentation, we have to find the correct front side of the object in order to generate consistent order of control points. To make sure the front side of the object is facing us, we have to do a PCA on 3D mesh and remember the direction that z axis has greatest value, and then do the PCA agiain on xy plane only for all mesh vertices. The direction of the principal component has to be aligned with our saved direction, otherwise we rotate the 3D mesh 90 degrees or 180 degrees to make the front side consistent over all objects in the same class. In addition, we have to guarantee the rotation matrix constructed by xy plane PCA and z axis satisfying the right hand rule such that z axis component equals 1 in case of any unwanted flipping to the model.

It is also very important that the predicted order of corners is in consistency with the order of our labels. We check the correctness of the ordering as one of our error metrics during testing. For some objects that may not have an obvious front side, for example table in square shape, we may allow a rotation in xy plane for the predicted corners and select the closest to the ground truth corners as the one to calculate loss.

Data filtering Our model aims to do camera localization in any indoor environments as long as there is same or similar object in the image as

the model has learnt in training procedure. While synthetic dataset can provide exactly same object in different scenes, which may result in a more accurate prediction, real data do not have exactly same object. We hope our model can recognize object in same class with any texture, but we filter out erroneous labels and object that has very different shape with the typical shape of the objects in the class.

Firstly, due to incomplete reconstruction, we may have erroneous labels that passing through the object, or there may be object in very different shape that may increase the difficulty for our network to converge. Thus, we need to filter out those very different 3D models. One reliable method is to compare the 3D meshes in grid granularity, and set a threshold to filter out the low similarity meshes. However, this method may result in big computation offset and also hard to deal with those slight offset cases for a thin plane, for example a table plane in slightly different height may result in very low grid similarity. To generate less computation overhead, we choose to filter out these cases by comparing the size of the bounding box. If the ratio between any side of the length of the bounding box with the bounding box of our chosen object model per class is out of the threshold range, we throw those data away. In this way, it is much easier to compute and is also able to filter out those incomplete reconstruction and those objects in very different shape.

Secondly, we also need to filter out those objects that are at the edge of the image. We filter out those images which have center of the object out of the image, and we set that more than 4 among the 8 corners should be within the range of image.

Thirdly, we check whether the object is occluded or not in the image. In ScanNet dataset, instance segmentation of each image is provided, and in SunCG, instance labeled images can also be rendered together with the RGB images. Thus, we can check the occlusion simply by comparing the 2D projections of all the vertices in the 3D object mesh with the 2D instance segmentation. We filter out those data that have more than 40% of vertices occluded in the image.

Lastly, there are sometimes invalid camera poses provided or the cases that the 2D projections of the object in the back of the camera, which are also the cases that we have filtered out for our training and testing data.

We provide our codes to automatically generate the labels of all objects and automatically filter out valid training and testing data for future work.

Chapter 5

Experiments and Results

In this chapter, we present experiments we did in this work and some valuable results we have achieved. We did experiments in both synthetic data and real data. We started with synthetic data to check how well our model can predict camera location in synthetic data on exactly same objects. It turns out the network can predict quite accurately for synthetic data. Then we did experiments in real data. We hope that by mixing synthetic data together with real data as training data, we can improve the result of prediction in real data.

5.1 Synthetic Data

We first did experiments on multiple views of a single object generated by SIXD Toolkit. We use objects from SunCG, and we generate multiple views of the object in random texture, with random translation within the camera's field of view, and with random distance to the object. We are also training with data in random background, testing in both no background and random background. As a result, the prediction is very accurate. Taking a 5 px threshold, the accuracy that the error of 2D projection of all vertices of the object 3D mesh between prediction and ground truth within the threshold is greater than 70%. Figure 51 shows the predictions on the test data of our generated synthetic data with no background. The red bounding box is the ground truth, the yellow box is the predicted 2D locations of the projected bounding box, the blue box is the reprojected bounding box using the camera pose computed from ground truth 2D locations i.e. the red box corners and given 3D control points locations, and finally the green box is the reprojected box using the camera pose computed from predicted 2D locations i.e. the yellow box corners and given 3D control points locations.

Then we did experiments on a whole synthetic image. We use SunCG to render the whole synthetic indoor environment images. Specifically, we

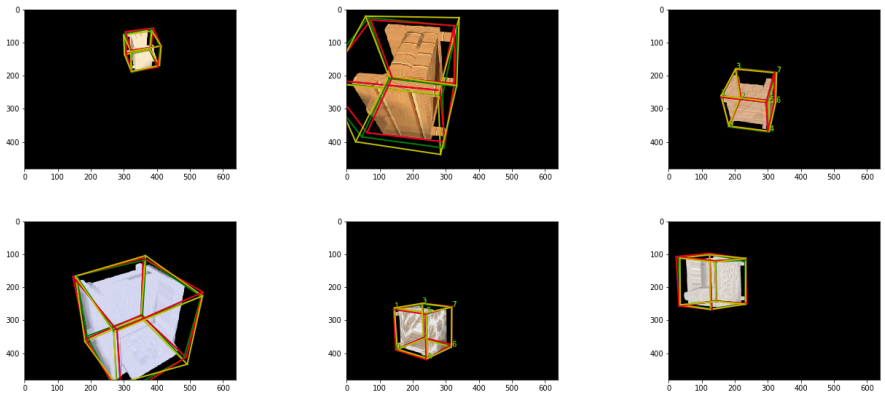


Figure 51: Examples of results of sofa object in multiple views generated with SIXD Tookit

filter out scenes containing our target object - sofa, render images of the scenes, and filter out images containing the object as we explained in section 4.1.3.

5.2 Real Data

For real data, we still have a lot of space for improvement. We will explain in more details in next chapter. However, we already achieve some good results on real data for single object camera pose detection. We get our training and testing data from 264 different scenes and in total 565 scans. Here are some results on Scannet Dataset.

Our model performs very well on training data or data from the same scene. When test on new scenes, which the network has never seen in the training period, we also get some quite good predictions. Figure 53 shows some examples of the good results we get on couch object in ScanNet test scenes. The red bounding boxes are the ground truth and the yellow bounding boxes are the predicted 2D points locations. As we can see, our model is robust to occlusion and different sizes of the object. Sometimes our ground truth label is not accurate due to the reasons we have explained in last chapter, and the predictions can even fix the noise and make a better bounding box than our labels. For example our ground truth label may not be upright due to noises in camera pose or incomplete construction of the object model, and the predicted bounding box can be upright and fit the object better.

Figure 54 shows some examples on different predictions with the ground truth, but they are not wrong. In this experiment, we try to filter out those images containing multiple same category objects. However, there are some noises maintain due to incomplete construction or objects in different sizes.

Figure 52: Examples of results of sofa object in SunCG test dataset

As we mentioned before, we filter out those non-similar object model according to the difference of the size of its bounding box with our chosen 3D model. And when we choose images containing single target category object, we check that weather there are any other objects in the same category after filtering out similar objects. This means there may still be some other

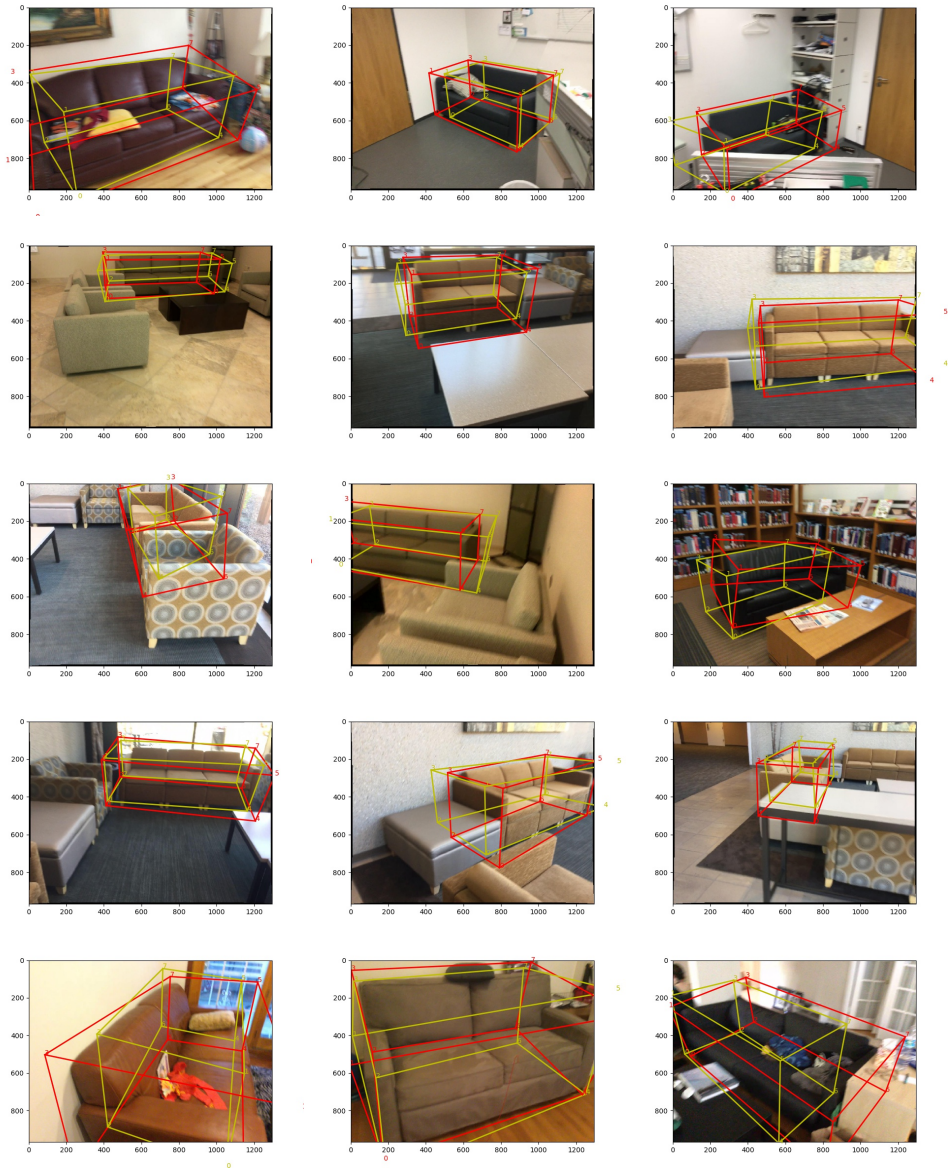


Figure 53: Examples of some good results of couch object in ScanNet test scenes

objects in the same category maintain in the image but with a different size i.e. out of the range of the threshold we set for bounding box size difference. Thus, when we test on these images, there may be other object predicted instead of the labeled one. As we can see, these predictions are not false, and should not contribute to our error measurements. Some of them predicts the other object in the image in the same category. Some of them try to fit

in with a quite different shape which they have never seen during training period. And some of them may detected another category object but the appearance is very similar to our target object, which are sometimes even hard to distinguish for human beings. For example sometimes the network regards the pillow on the couch as the back of another small couch, which are quite reasonable predictions.

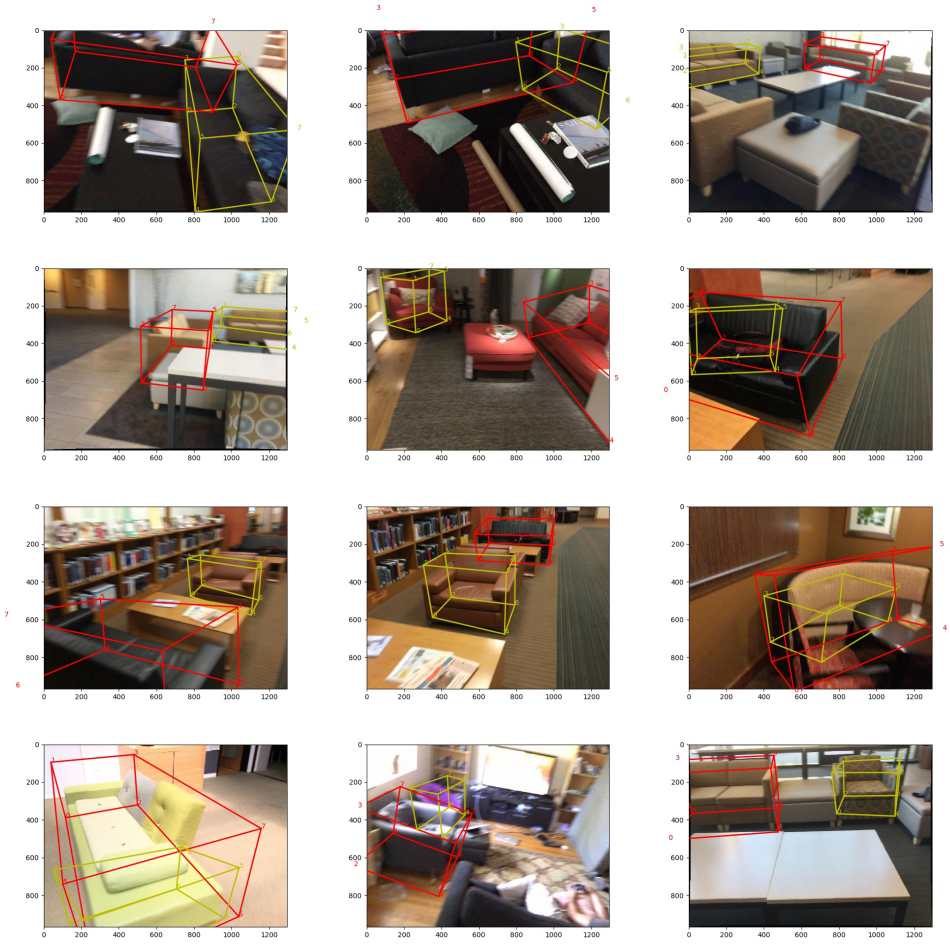


Figure 54: Examples of some reasonable results we of couch object in ScanNet test scenes

There are also a lot of failed cases. Figure 55 shows these incorrect predictions. Consider the reasoning, there are less accuracy for predictions from the back of the couch. This is also reasonable for human beings, as it's a lack of important features when only the back of the couch is present in the image. The predictions are poorer for the objects having similar texture or color with the background, which is sometimes also difficult to distinguish

in human eyes. The predictions are also poorer for the object containing different textures or colors itself. This may make the network partition the object and predict only part of the object. For example, the network has never seen a couch having different colors in its back, or couch with no back in some part, and the model only predict a part of the couch which contains a back and consists of only one color/texture. And finally, those object in quite special shapes that the network has never seen during training result in poor predictions. Of course our training data size is not big enough. By enlarging the size of the training dataset, our model may generalize better to more different shapes of the objects in the same category.

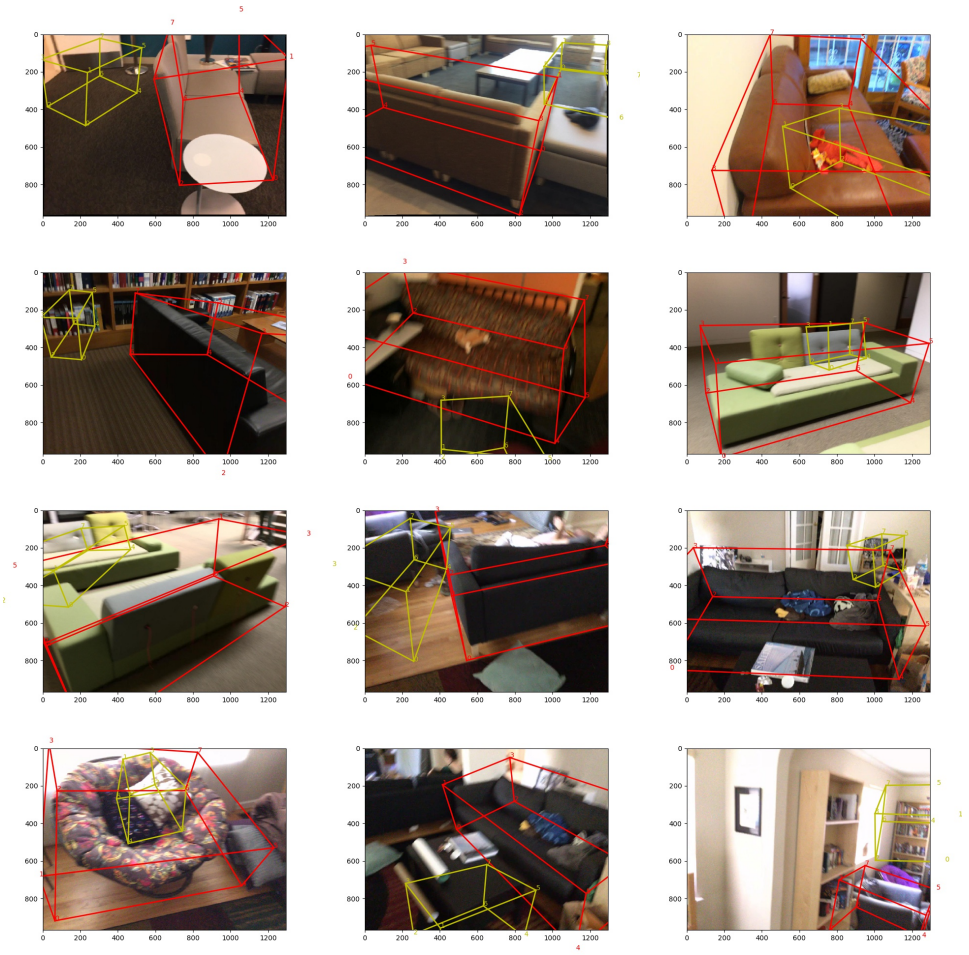


Figure 55: Examples of some failed results of couch object in ScanNet test scenes

Figure 56: Examples of the some good results of chair object in ScanNet test scenes

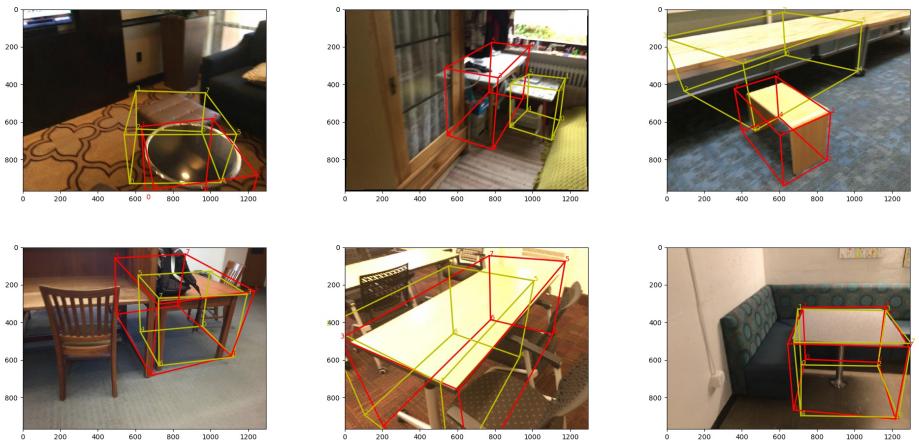


Figure 57: Examples of the some good results of table object in ScanNet test scenes

Chapter 6

Limitations and Future Work

There are a lot of limitations on our current work and a lot of potential work in the future. In this chapter, we summarize the limitations in our work from our model to the datasets, and state possible future work to get a better result in real-time image-based camera localization.

6.1 Network

Since our model learns to predict 2D projections of 3D bounding box from limited training data, sometimes it cannot generalize well to objects in new or unusually perspective of views. For example, in our testing data for couch, the result of predicting from back of the couch is not as good as the prediction from the front view.

Our loss function also does not distinguish the error of predicted 2D points and ground truth in small 3D bounding box versus large 3D bounding box. A small error in a large box is generally negligible but the same error in a small box has a much greater effect on the result or IOU. Thus, further improvement on the loss function is possible.

6.2 Datasets

6.2.1 SIXD

Although SIXD tollkit can help us generate quite a lot different views of the object in random textures, the views we generated are still not realistic enough.

To adress these limitations, one of the future work we can do is to adjust the initial orientation of the 3D object model so that most of the camera views have a fixed upright direction aligned to z=1. Also, automatically filtering

out some unrealistic views such as the view from the bottom of the object is also necessary. Since in real indoor environment, we rarely see the furniture from a view to the bottom of it. The random background utilized should better be of indoor environments containing recognizable furnitures. And if possible, it would be great if the floor can be detected and put the object onto floor level so that it is not obviously floating in the air.

6.2.2 SunCG

random texture, random x scale other objects

predict multiple classes

Example Paragraph Dummy text.

Example Subparagraph Dummy text.

3D model similar to our specified model

intrinsic parameter

Bibliography

[1] Eric Brachmann, Alexander Krull, Sebastian Nowozin, Jamie Shotton, Frank Michel, Stefan Gumhold, and Carsten Rother. Dsac-differentiable ransac for camera localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6684–6692, 2017.

[2] Ben Glocker, Shahram Izadi, Jamie Shotton, and Antonio Criminisi. Real-time rgb-d camera relocalization. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 173–179. IEEE, 2013.

[3] Wadim Kehl, Fabian Manhardt, Federico Tombari, Slobodan Ilic, and Nassir Navab. Ssd-6d: Making rgb-based 3d detection and 6d pose estimation great again. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1521–1529, 2017.

[4] Vincent Lepetit, Francesc Moreno-Noguer, and Pascal Fua. Epnp: An accurate o (n) solution to the pnp problem. *International journal of computer vision*, 81(2):155, 2009.

[5] Pushmeet Kohli Nathan Silberman, Derek Hoiem and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *ECCV*, 2012.

[6] Mahdi Rad and Vincent Lepetit. Bb8: A scalable, accurate, robust to partial occlusion method for predicting the 3d poses of challenging objects without using depth. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3828–3836, 2017.

[7] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[8] Jamie Shotton, Ben Glocker, Christopher Zach, Shahram Izadi, Antonio Criminisi, and Andrew Fitzgibbon. Scene coordinate regression forests for camera relocalization in rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2930–2937, 2013.

[9] Bugra Tekin, Sudipta N Sinha, and Pascal Fua. Real-time seamless single shot 6d object pose prediction. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 292–301, 2018.

[10] Jian Wu, Liwei Ma, and Xiaolin Hu. Delving deeper into convolutional neural networks for camera relocalization. In *2017 IEEE International Conference on Robotics and Automation (ICRA)*, pages 5644–5651. IEEE, 2017.

[11] Yihong Wu, Fulin Tang, and Heping Li. Image-based camera localization: an overview. *Visual Computing for Industry, Biomedicine, and Art*, 1(1):1–13, 2018.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Declaration of originality

The signed declaration of originality is a component of every semester paper, Bachelor's thesis, Master's thesis and any other degree paper undertaken during the course of studies, including the respective electronic versions.

Lecturers may also require a declaration of originality for other written papers compiled for their courses.

I hereby confirm that I am the sole author of the written work here enclosed and that I have compiled it in my own words. Parts excepted are corrections of form and content by the supervisor.

Title of work (in block letters):

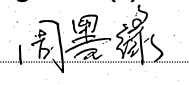
Authored by (in block letters):
For papers written by groups the names of all authors are required.

Name(s):	**First name(s):**

With my signature I confirm that

- I have committed none of the forms of plagiarism described in the 'Citation etiquette' information sheet.
- I have documented all methods, data and processes truthfully.
- I have not manipulated any data.
- I have mentioned all persons who were significant facilitators of the work.

I am aware that the work may be screened electronically for plagiarism.

Place, date	**Signature(s)**
	

For papers written by groups the names of all authors are required. Their signatures collectively guarantee the entire content of the written paper.