Universität Hamburg
Department Informatik
Knowledge Technology, WTM

Advanced Vision Architectures: Hourglass Networks and CornerNet

Seminar Paper

Neural Networks

Jan-Gerrit Habekost, Marcus Rottschäfer

Matr.Nr. 7323666,7327475

9habekos@informatik.uni-hamburg.de
9rottsch@informatik.uni-hamburg.de

09.07.2020

Abstract

The current developements in Convolutional Neural Networks (CNNs), together with the availability of massive amounts of labeled data and computational possibilities, led to a paradigm shift in Computer Vision. This shift also heavily influenced object detection, where Deep CNNs achieve competitive results on benchmark datasets like COCO. In the subclass of one-shot detectors, object detectors which rely on a single pass through the network, anchor box approaches recently marked the state-of-the-art. However, the choice of anchor boxes is heavily influencing the networks performance and is often a difficult task in terms of choosing suitable hyperparameters for the one-shot detectors.

To overcome the hyperparameter choices, inherent in anchor box approaches, Law et al. proposed the CornerNet CNN, which tackles the task of object detection as a keypoint-estimation problem. Relying on a stacked-hourglass architecture, the network predicts the upper-left and bottom-right coordinate for an object bounding box. In this paper, we train a fast successor called CornerNet-Squeeze on an object tracking dataset called NICO-OI, which is a collection of labeled videos of robot-object interactions. Our results show that CornerNet-Squeeze can successfully be trained on the NICO-OI dataset for object detection, reaching an AP score of 0.9. Further, we observe that the network nicely deals with occlusions and in-plane rotations, partly memorizing object shapes and inferencing about object positions. Our findings let us conclude that CornerNet-Squeeze can be applied on the NICO robot for object detection.

Contents

List of Abbreviations 1

1 Introduction 2

2 Related Work 3

3 Background 4
- 3.1 Hourglass Architecture 4
- 3.2 Prediction Module 5
- 3.3 CornerNet-Squeeze 7

4 Implementation 8
- 4.1 NICO-OI Dataset 8
- 4.2 Training 9

5 Results 10
- 5.1 Discussion 11

6 Conclusion 12

7 Outlook **13**

Bibliography **14**

Abbreviations

AP	Average Precision
CNN	Convolutional Neural Network
IoU	Intersection over Union

1 Introduction

In the field of mobile and humanoid robotics, computer vision is key for understanding of the robots surrounding. It is a well studied, yet not fully researched field with a high number of upcoming ideas and approaches. One approach that is utilized quite often is object detection with Anchor Boxes. These boxes are rectangles that are drawn around detected objects and intersect their extreme points. While Anchor Boxes achieve remarkable results when it comes to accuracy, they also come with a high computational effort, as several thousand candidates for the boxes must be calculated. This makes application for real-time tasks complex and alternatives to Anchor Boxes were researched. CornerNet presents a novel approach for Object detection by keypoints. We will present how the reduction of computational cost makes it possible to apply CornerNet for detecting objects with the *NICO* robot.

The CornerNet Architecture by Law and Deng [7] showed remarkable results compared to other keypoint-based methods. It accomplished to overcome other widely used and researched approaches, such as YOLO-v3, in detection quality. However, the downside of the CornerNet architecture lies in its efficiency, as it takes a comparably long time for the processing of an image. Therefore, in it's original form, it is likely not applicable to time critical systems, as the practitioner needs to tune down the resolution of the input images, which leads to a significant loss in its accuracy. The extensions CornerNet-Saccade and CornerNet-Squeeze aim to overcome these problems. [8]

The Stacked Hourglass Architecture proposed by Newell et. al. [11] performed well for the estimation of human poses from images. The method is related to encoder-decoder and conv-deconv approaches for feature extraction. The core element of the approach lies in the concatenation of multiple hourglass modules, separated by fully connected convolutional layers. A hourglass module thereby consists of a convolutional part reducing features over different scales, based on max-pooling operations and an upsampling part. The convolutional layers of the upsampling part mirror the quadratic sized layers of the convolution part to combine and populate back, features over different scales onto the original input size, resulting in an hourglass-form shaped module. A more detailed explanation of the hourglass architecture can be found in section 3.1, a visualization is given in Fig. 2.

With introducing CornerNet-Lite, Law et. al. [8] tried to overcome the computational boundaries of CornerNet, while aiming to maintain a similar level of accuracy. This lead to the extend of the initial CornerNet architecture into two new approaches, CornerNet-Saccade and CornerNet-Squeeze. CornerNet-Saccade contains a bioinspired mechanism to generate attention maps for images, neatly related to a phenomenon observed in human text-reading, which is embodied in a rapid eye movement between fixation points. Therefore it is possible to focus on subsets of the images that are fed into the network. In contrast CornerNet-Squeeze adapts ideas of SqueezeNet [6] and MobileNets [5], which raise additional requirements for the convolutional layers in the network architecture. In their work, Law et. al. raised the hypothesis, that a combination of CornerNet-Saccade and CornerNet-Squeeze would result in the best performance. The team validated all their architectures on a Desktop-PC with an i7-7700k CPU and a GTX 1080TI GPU. In their results they state that the CornerNet-Squeeze architecture outperformed CornerNet, CornerNet-Saccade and most remarkable the state-of-the-art and widely spread Yolo-v3 architecture, as well as the combination of CornerNet-Saccade and CornerNet-Squeeze in inference time. [8]

The CornerNet-Squeeze architecture has an inference time of 30ms per image, which claims to make it utilizable for online-object detection tasks, as a normal camera stream provides 30 frames per second and the computation time for 1 second of video stream is then bound to 900ms, assuming the hardware setup that is described in the paragraph above. Based on these findings, in this work we will seek to identify potential application domains for online object detection tasks, in the context of robotic laboratory environments and consecuently evaluate the performance of CornerNet-Squeeze onto these.

2 Related Work

In the computer vision task of object detection with CNNs, mainly two approaches have been popularized in the recent years, namely the two-stage detectors and the one-stage detectors.

The former approach partitions the object detection into two separate tasks. At first, a module processes the image and proposes a set of Region of Interests (RoIs). These RoIs are proposed for example with a Selective Search algorithm (R-CNN [3], Fast R-CNN [2]) or with a neural network like a region proposal network (Faster R-CNN [15]). In the second stage, these RoIs are further processed by a CNN to generate object category and bounding box predictions. Two-stage detectors often offer competitive performance in object detection. However, they are typically more computationally expensive than the one-stage detectors and may therefore not be used in some constrained applications like embedded systems, where efficiency is important [8].

One-stage object detectors, on the other hand, aim to solve the task directly without using a RoI pooling step, which allows the object detection in a single network, in exchange suffering in accuracy compared to two-stage approaches. One-stage detectors often make use of anchor boxes, which are boxes of various size and shape containing possible detection candidates [7]. These anchor boxes are used among a variety of successful one-shot detectors like SSD [10], DSSD [1], the YOLO approaches (YOLO9000 [13], YOLOv3 [14]) or RetinaNet [9], which for the first time achieved a performance superior to two-stage approaches of that time. Typically, anchor boxes are placed densely over an image, with the one-stage detector scoring each box and possibly refining its position to best match an object detection [7]. So, in order for the object detection to work well, a network requires a large amount of anchor box candidates (e.g. over 100.000 anchor boxes in RetinaNet [9]). The choice of anchor boxes heavily influences the performance of the network and is therefore an important hyperparameter for the object detector.

3 Background

In search of one-stage detectors overcoming the necessary hyperparameter choices inherent in anchor box approaches, Law et al. proposed the CornerNet object detection framework [7, 8]. CornerNet [7] and its successors CornerNet-Saccade and CornerNet-Squeeze [8], together called CornerNet-Lite, implement a novel approach for object detection, based on keypoint estimation.

The CornerNet approach completely overcomes the need for anchor boxes, by predicting corner heatmaps for paired keypoints, which represent the top-left and bottom-right corner of the object's anchor box. The anchor box can, as a result, be recovered from the predicted keypoints. The authors state that the reason for the reduction of complexity lies in the fact that for each keypoint, the search space is bound to $O(wh)$, where w is the width and h the height of the input image. [7]

The overall architecture is shown in Fig. 1. On a high level, CornerNet consists of two parts: The Hourglass backbone and the two prediction modules. There is one prediction module per keypoint. In the first step, the hourglass network performs multi-scale feature extraction. The prediction modules take the output of the Hourglass Network as an input to predict a heatmap for the corner, a shortest distance embedding to link the two keypoints and an offset to handle uncertainties, when it comes to pixel-to-pixel ground truth matching. [7]

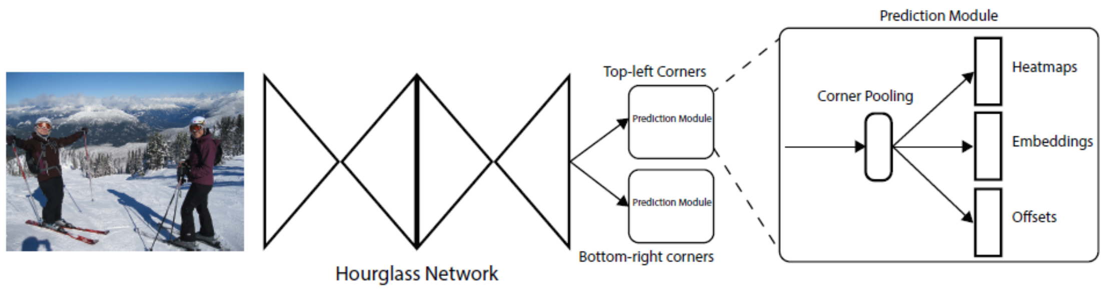


Figure 1: CornerNet adapts the Hourglass architecture as a backbone to feed multi-scale features into two prediction modules. These output predictions for corner heatmaps, embeddings for keypoint pairing and offsets to deal with smaller deviations. Source: [7]

3.1 Hourglass Architecture

The Hourglass architecture introduced by Newell et. al. [11] builds upon the idea to down-sample the input, with a group of convolutional layers that are ordered in a decreasing size, in the first step. This makes it possible to extract features on different scales. The up-sampling part of an Hourglass module then aims to back-project the extracted features back onto the original image size. To pass features from different scales from the down-sampling to the up-sampling process, by-pass layers are used. Each convolutional layer in the down-sampling part of the module has an counterpart deconvolutional layer in the up-sampling part, which it is linked to by exactly one by-pass layer. A visualization is shown in fig. 2. To increase the quality and observe higher level features, Hourglass modules can be stacked, so features get reprocessed multiple times.

The Hourglass modules of CornerNet are slightly customized to provide better prerequisites for the object detection task. In the work of Newell et. al. [11] max-pooling layers are used for down-sampling, CornerNet uses stride 2 filters instead. The total number of feature reduction (down-sampling) layers is 5. The by-pass layers contain two residual blocks, that encapsulate the features for a specific feature-scale. To populate the features to a higher scale, the features from the bypass layers are added to the output of the prior up-sampling layer using nearest-neighbour up-sampling. Therefore, the resulting layer has the size of the observed by-pass layer and features from lower scale layers are fused with those of the residual block. [7]

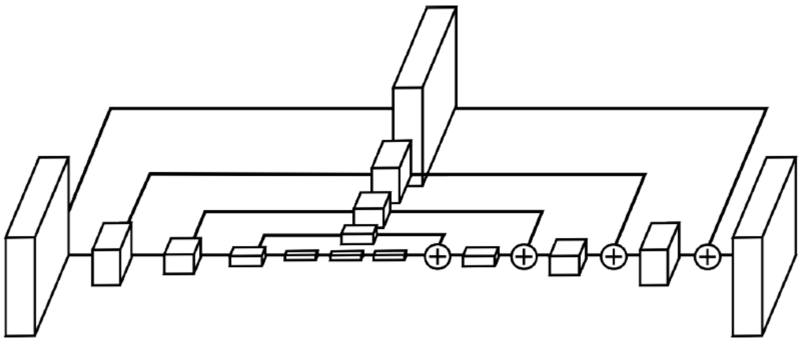
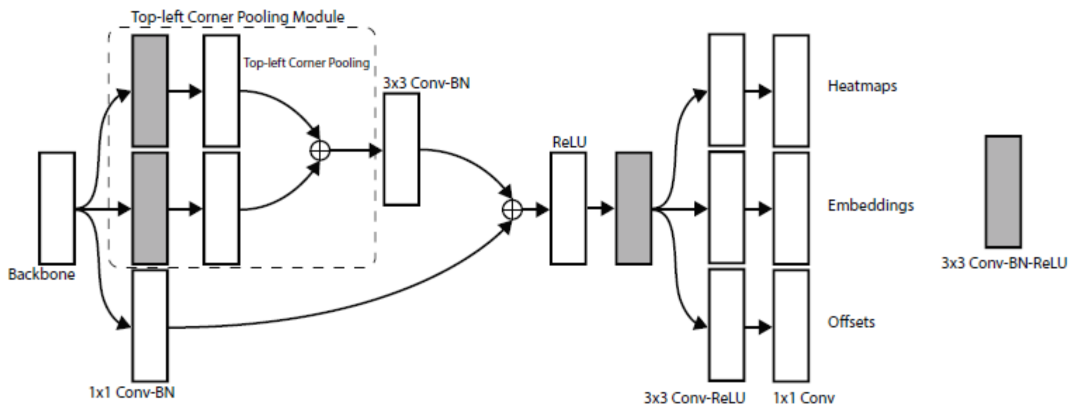


Figure 2: The Figure obtained from Newell et. al. [11] visualizes the by-pass mechanism to pass detected features from the down-sampling process to the up-sampling process.

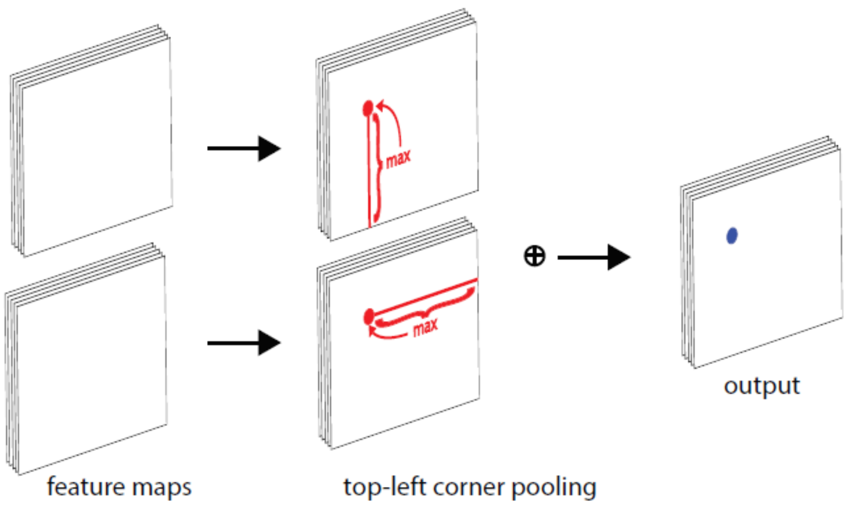
To mitigate the loss of information, as already mentioned, Hourglass modules can be stacked. To further support the reprocessing of features, in CornerNet, a 1x1 Batch Normalization convolutional layer is applied to both, input and output of an Hourglass module. The results are added up and build the actual input for the next Hourglass module [7].

3.2 Prediction Module

In order to predict the keypoints of an object's bounding box, Law et. al. propose a prediction module architecture that outputs a *heatmap* for the correct corner position, an *embedding* that sets the corner in context with it's pair keypoint and *offsets* to mitigate impreciseness in CornerNet's predictions. There is one prediction module for each of the two keypoints, whose structures are equal. The structure of the prediction modules is shown in Fig. 3a. The input for the prediction module is the output of the hourglass backbone. Following from this, the dimensions are equal to the dimensions of the image that is fed into CornerNet [7].



(a) The figure shows the overall network structure of the prediction module. It takes the output of the hourglass backbone as input. In the next step two corner pooling layers are applied in parallel (one horizontal and one vertical). The output is added up with the output of a residual block and processed by two ReLU layers. Followed by this, the network is split into three branches, for heatmaps, embeddings and offsets



(b) The figure shows the process within the corner pooling module (dotted part of fig. 3a). The max-pooling operation is applied twice onto the feature maps, once horizontally (from right to left) and once vertically (from bottom to top).

Figure 3: The two figures obtained from Law et. al. [7] visualize the prediction module of CornerNet. Fig. 3a shows the network structure of the prediction module. Fig. 3b shows the custom max-pooling operation within the corner-pooling layer, which is utilized to predict the bounding box corners from feature maps.

The main components of the prediction module are corner pooling layers. These layers perform custom max-pooling operations to locate corner positions. Since the bounding box's corners are in most cases located out of the actual object area, this approach aims to identify the object's extreme points from the given feature maps and construct the corners from these.

In the following we will describe the method for the top-left corner pooling, the method for the bottom-right corner pooling works similar. A visualization for the corner pooling method is given in Fig. 3b. The custom max-pooling operation is performed once horizontally and once vertically. In the horizontal layer, the algorithm runs through the rows of the feature map, from right to left. The yet highest observed value builds the threshold. If the values of the fields located left of the discovered threshold are lower than the threshold, they are replaced by the threshold. If the value of the currently observed field is higher than the threshold, it is accepted as the new threshold. The vertical pooling layer works equal, except that the algorithm runs through the columns of the feature maps, from the bottom to the top. Before the hourglass output is fed into each of the two pooling layers, it is normalized and down-sampled through a 3x3 convolutional batch-normalization ReLU layer. The output of both pooling layers is added up and builds the overall output of the corner pooling module [7].

The corner pooling output is further down-sampled by a 3x3 convolutional batch normalization layer and afterwards added up with a batch normalized (1x1 Conv-BN) residual instance of the hourglass output. Since both instances have different dimensions, the down-sampled corner pooling output is up-sampled to the dimensions of the residual instance, through the projection shortcut function. The projection shortcut performs identity mapping and fills the differences in dimensions with zero entries. The zero entries form a padding around the correctly mapped entries. This way, the low-level corner features are set into context with the global features found by the hourglass module. The result again has the same dimensions as the original image that is to be classified and the output of the hourglass network. The result is fed into a ReLU layer, followed by another 3x3 convolutional batch-normaliaztion layer. The output is populated to three different network branches for the heatmaps, embeddings and *offset*. All three branches have the same structure: They consist of a 3x3 covolutional ReLU down-sampling layer, followed by a 1x1 convolutional layer. [7]

3.3 CornerNet-Squeeze

As mentioned in the introduction (Section 1), in this work, we focused our research on CornerNet-Squeeze, due to it's short inference-time and the ability to apply it to real-time object detection scenarios. For CornerNet-Squeeze, the architecture of CornerNet is modified, based on ideas from SqueezeNet [6] and MobileNets[5]. We will not characterize SqueezeNet and MobileNets in detail at this point, but rather give a high-level description of the modifications in the architecture of CornerNet Squeeze. For further details and results, see the underlying publications of this work. [8]

The first idea adapted from SqueezeNet is the fire-module. It consists of a 1x1 convolutional layer, which is thought to reduce the number of channels in the feature-maps and therefore is also referred to as Squeeze-Layer. It is followed by an Expanding-Layer. This layer consists of a 1x1 convolutional filter combined with a depth-wise applied 3x3 ReLU kernel. The fire modules replace the 3x3 convolutional by-pass elements in the hourglass module, described in Section 3.1 and which are shown in Fig.2 [8].

One other idea of SqueezeNet is to delay the application of down-sampling layers as long as possible during the classification process. Inspired by this, since the hourglass architecture makes it impossible to directly apply the idea, one down-sampling layer and it's corresponding up-sampling layer, were removed from the hourglass architecture. Following from this, while CornerNet has 4 down-sampling layers in each hourglass, for CornerNet-Squeeze the number of downsampling layers is 3. To mitigate this difference in dimensions, the input image is down-sampled 3 times before being fed into the hourglass network, while in the traditional CornerNet this is only done 2 times. Also, the nearest-neighbour up-sampling algorithm, which is applied to the output of the hourglass, is replaced by a 4x4 transpose convolutional layer. [8]

Additionally, the 3x3 convolutional filters in the prediction module are simply replaced with 1x1 filters.

4 Implementation

In our implementation, we evaluate the performance of the Hourglass architecture-based CornerNet-Lite framework on an object-tracking dataset called NICO-object interaction (NICO-OI) [4]. In section 4.1, the dataset is described in more detail.

For our implementation, we train CornerNet-Squeeze on the NICO-OI dataset. We provide a detailed evaluation on the performance of the network on this object tracking task, aiming at creating an object detection and tracking framework applicable to the real NICO robot.

4.1 NICO-OI Dataset

The NICO-object interaction dataset [4] comprises 60 videos of robot-object interactions captured from the NICO (Neuro-Inspired COmpanion) humanoid developmental robot. NICO recorded 60 hand-object interactions, from the perspective of its eyes, with 30 frames per second in a RGB video stream. The interactions include push, pull, grasp and lift actions on a variety of objects like dice, plastic fruits, handkerchiefs and other toy objects [4]. Figure 4 shows two frames taken from the dataset depicting the robot in performing a grasp and lift, with the manually labeled bounding boxes rendered in the image. The dataset comprises around 40k labeled frames. In total, there are 22 different classes of objects, for example heavy_banana and blue_car in the example frames in Fig.4. Due to severe occlusions from the hand, the effect of object manipulation is in some cases difficult to observe visually. As the authors state, this object-tracking dataset provides explicit difficulty in the object scale and in-plane rotation [4], as the object is often rotated and moved towards the robots camera.

The dataset has been chosen with the intent to determine whether the CornerNet-Lite architecture can be used on the NICO robot for object detection tasks. As

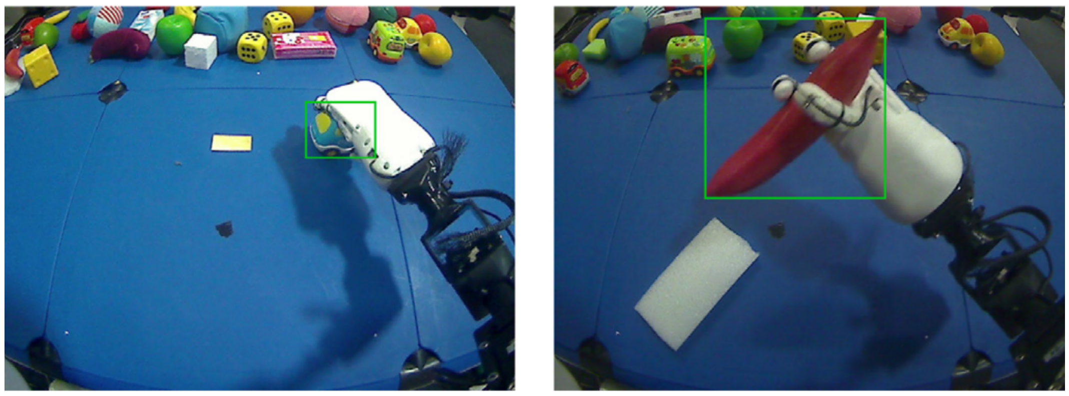


Figure 4: Two example frames from the NICO-OI dataset, with the ground-truth label bounding box drawn in green. Taken from [4]

the NICO-OI dataset provides a set of labeled videos of NICO interacting with objects, it can be used to train and fine tune CornerNet-Lite on the specifics of the NICO robot. Evaluating the object detection performance on a subset of NICO-OI therefore allows us to make assumptions about the performance of CornerNet-Lite on the NICO robot.

4.2 Training

For the training, we chose the CornerNet-Squeeze architecture. With the intention of running the network on the NICO robot, CornerNet-Squeeze appeared as the best choice because of its compact backbone architecture and real-time inference capabilities [8].

A random 70%-15%-15% split was done in order to get the training, validation and test dataset respectively. The training took place on a GPU Server with two NVIDIA GTX 1080, allowing for an effective batch size of 24. Many of the hyperparameter choices were met to comply with the CornerNet-Squeeze network trained on the COCO dataset by Law et. al [8]. We trained for a total of 100.000 iterations. Additionally, a model with 500.000 iterations has been trained, which took about 125h, but due to inconsistencies we are not able to reliably provide its performance here. The neural network was implemented in PyTorch [12], following the implementation provided by CornerNet-Squeeze [8]. To allow the training on the COCO API-based implementation, we additionally converted the NICO-OI dataset into a format corresponding to COCO.

Figure 5 depicts the development of the training- and validation loss for training CornerNet-Squeeze on the NICO-OI dataset for 100.000 iterations. Same as the original CornerNet-Lite, we used Focal loss [9] as our training minimization objective. Focal loss is a loss metric well suited for one-shot object detectors, putting emphasis on hard training examples so the network is not overwhelmed by its performance on more easy training images [9].

The validation loss development shows that at the current setting of 100.000 iterations, there is no overfitting happening yet. We conclude so because the validation loss does not succeed the training loss (with the exception of a few iterations) during the training. Although there is a slightly visible increase in validation loss in the end, we interpret this development in that further training can be done. However, due to time and resource constraints, we report the performance for a total of 100.000 iterations here.

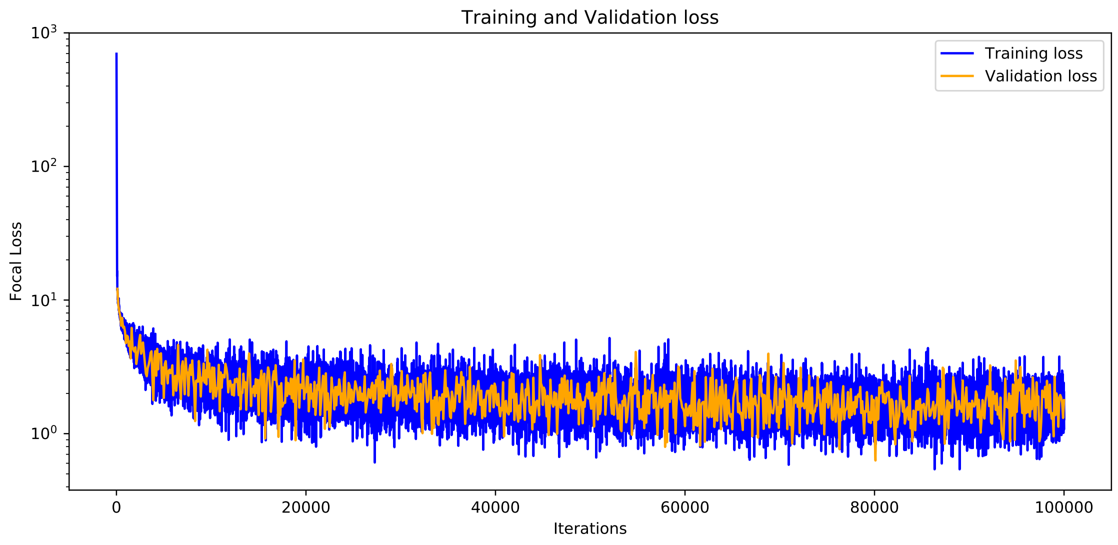


Figure 5: Training- and validation loss evaluated after each fifth- and hundredth iteration, respectively. The Focal loss metric has been applied [9] and provides the minimization target.

5 Results

We evaluated the model with regard to the Average Precision (AP) for 100 snapshots, exported at every 1000 iterations during training. We found CornerNet-Squeeze to provide the best model at about 100,000 iterations. Figure 6 shows the development of the test set performance for different numbers of iterations, ranging from 1000 to 100,000. We observed that the test set performance rapidly increases in the beginning, reaching 0.83 AP after just 20.000 iterations. The best AP score is reached at about 100.000 iterations (0.90). We think that CornerNet-Squeeze did not reach it's maximum performance in the executed 100,000 training steps. However, as reported in the previous section, further training could be applied to increase the performance even more.

The AP score we used for the model evaluation was equal to the COCO evalutation metric. This is the AP at IoU=0.50 : 0.95 score, the primary metric of the COCO dataset evaluation. This metric measures the area under the Precision-Recall curve, interpolated with a 101-point measure, averaged over 22 classes, averaged over 10 Intersection over Union (IoU) levels in the range of 0.5 to 0.95 in 0.05 steps. AP is always in the range of $[0, 1]$. IoU measures the quality of the

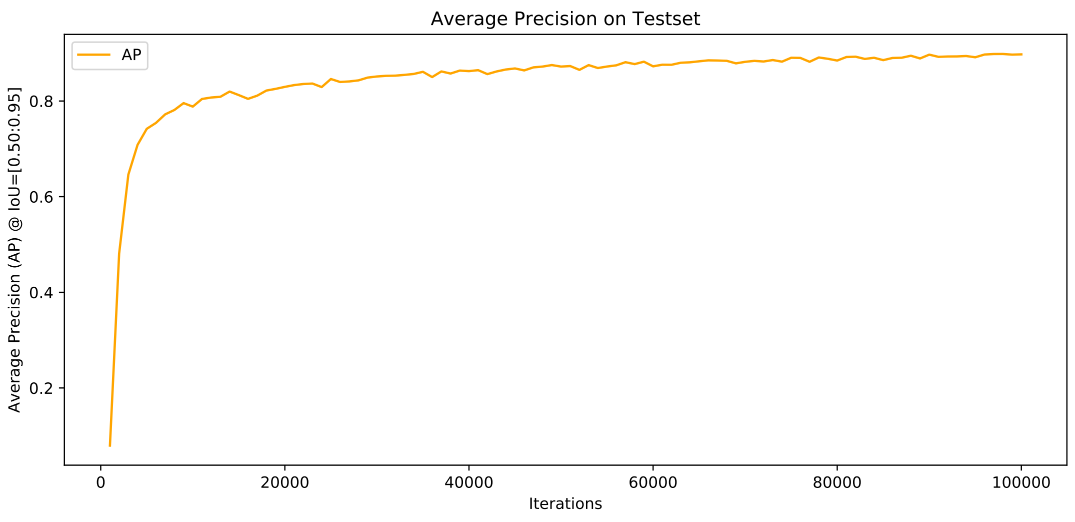


Figure 6: Development of the AP score for the test set on different numbers of training iterations.

bounding box prediction. It is the overlap area of the ground-truth bounding box and predicted bounding box, divided by the total area made up of the ground-truth and predicted bounding box. A IoU level of 0.5 means that a prediction is marked as correct, if the IoU score is at least 50%. To bring those results in contrast, CornerNet-Squeeze achieves an AP of 0.34 on the COCO dataset [8].

With a test set performance of AP=0.9 reported so far, we successfully trained CornerNet-Squeeze to perform object detection on the NICO-OI dataset. Although there are many similar images in terms of pose, lightning and contrast (same camera, same blue background, often similar object position), this dataset is not as easy as it might appear. As mentioned in the previous section, this is due to often severe occlusions from the robots hands, as well as in-plane object rotations, originating from the robot's interactions [4]. E.g. grasping an object and lifting it often turns it backwards in the move. Figure 7 shows two such difficult cases and the corresponding predictions made by the network. Both images are taken from the test set. In the left image, a blue car is successfully recognized and located, although the robots hand severely occludes the object, up to the point where only a small part of the object is visible.

In the right image, we observe that the network is also capable of detecting objects that are rotated and partly occluded. The red car was turned on its back, following a grasp and lift action, but still is detected by the network. Although handpicked, these examples show that the network is able to deal with the specific difficulties provided by the NICO-OI dataset.

5.1 Discussion

Since the objects, contained in the NICO-OI dataset, have similar dimensions with respect to the width and height of the bounding box, we hypothesise, that the

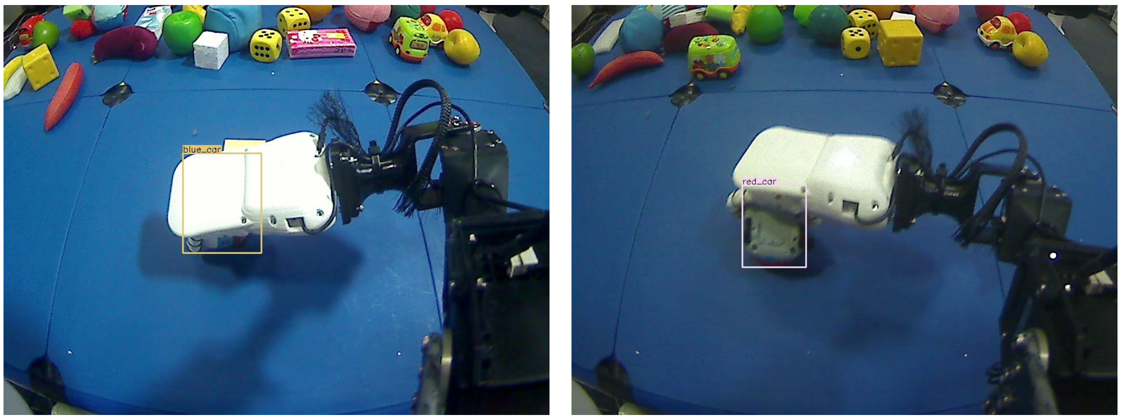


Figure 7: Handpicked examples of the object detection performance on two test set images. We observe that CornerNet-Squeeze is very robust to severe occlusions of the object (left) and also nicely deals with objects that have been rotated by the robot (right).

network learns the ratio of width to height, for each of the 22 objects. Therefore, it is able to recover the bounding box's height from the width it detects, when only a small part of the object area ($\approx\leq 10\%$) is visible. Often these situations occur, when NICO approaches a grasp on an object and only small parts protrude at the top or bottom of the robots hand. This is also visible in fig. 4 and fig. 7.

Obviously, one reason for the behavior described above, can be lead back to the properties of the given dataset, which support the training process and were already discussed prior. Besides, we further hypothesise, that the linearization of the bounding-box corner-detection problem in CornerNet's corner-pooling layers highly supports the network in learning a width-to-height ratio during training.

Besides CornerNet's precision of over 90% and it's robustness to hand occlusion, we think that a training with domain randomized and noizy samples could further improve the precision. Overall, we think that CornerNet-Squeeze is capable to perform well as the object-detection backbone of a robotic application situated in a laboratory environment.

6 Conclusion

We trained the novel CornerNet-Squeeze CNN, a keypoint-based one-shot object detector with a competitive performance and inference time, on the NICO-OI dataset. This dataset, originally made for object tracking tasks, has some difficulties due to occlusions and in-plane rotation [4]. The AP score of 0.9 reported on the test set so far lets us confidently conclude that CornerNet-Squeeze can successfully be trained on the NICO-OI dataset to perform object detection.

It should be noted that, due to real-time inference capabilities, we choose the

CornerNet-Squeeze architecture as the backbone for the object detection task on the NICO-robot. Even though it achieves astonishing performance on the detection task at hand, we hypothesise that this performance can be increased even further when using a different CornerNet architecture, such as CornerNet-Saccade [8]. This is based on the performance measured on the COCO-dataset, where CornerNet-Saccade achieves an AP score of 0.43, compared to 0.34 AP for CornerNet-Squeeze [8]. Further evaluation will be needed to confirm whether this performance increase on the COCO dataset can be transferred to the NICO-OI dataset as well.

Training- and validation loss development (Fig. 5) and the reported AP scores (Fig. 6) show that there is still more to be learned by the network. Even after 100k iterations, the network appears to not have reached its performance limit yet, and further training can be done to explore maximal performance.

7 Outlook

The CornerNet architecture and it's successors were already applied in different experiments and further offshoots of the CornerNet architecture variants have been developed. In 2019 Zhou et. al. [16] proposed ExtremeNet. It is based on the ideas of CornerNet, but tries to recover an objects bounding box from detected object extreme points. In addition it offers a module for semantic segmentation (DEXTR). From our perspective, it would be interesting to compare CornerNet-Squeeze and ExtremeNet in terms of inference time and precision. The capability of semantic segmentation would be a valuable extension of the good results of CornerNet-Squeeze.

References

[1] Cheng-Yang Fu, Wei Liu, Ananth Ranga, Ambrish Tyagi, and Alexander C Berg. Dssd: Deconvolutional single shot detector. *arXiv preprint arXiv:1701.06659*, 2017.

[2] Ross Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[3] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[4] Stefan Heinrich, Peer Springstübe, Tobias Knöppler, Matthias Kerzel, and Stefan Wermter. Continuous convolutional object tracking in developmental robot scenarios. *Neurocomputing*, 342:137–144, 2019.

[5] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. MobileNets. *arXiv preprint arXiv:1704.04861*, 2017.

[6] Forrest N. Iandola, Matthew W. Moskewicz, Khalid Ashraf, Song Han, William J. Dally, and Kurt Keutzer. SqueezeNet. *arXiv*, 2016.

[7] Hei Law and Jia Deng. Cornernet: Detecting objects as paired keypoints. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 734–750, 2018.

[8] Hei Law, Yun Teng, Olga Russakovsky, and Jia Deng. Cornernet-lite: Efficient keypoint based object detection. *arXiv preprint arXiv:1904.08900*, 2019.

[9] Tsung-Yi Lin, Priya Goyal, Ross Girshick, Kaiming He, and Piotr Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[10] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C Berg. Ssd: Single shot multibox detector. In *European conference on computer vision*, pages 21–37. Springer, 2016.

[11] Alejandro Newell, Kaiyu Yang, and Jia Deng. Stacked hourglass networks for human pose estimation. In *European conference on computer vision*, pages 483–499. Springer, 2016.

[12] Adam Paszke, Sam Gross, Soumith Chintala, Gregory Chanan, Edward Yang, Zachary DeVito, Zeming Lin, Alban Desmaison, Luca Antiga, and Adam Lerer. Automatic differentiation in PyTorch. 2017.

[13] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[14] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv preprint arXiv:1804.02767*, 2018.

[15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

[16] Xingyi Zhou, Jiacheng Zhuo, and Philipp Krähenbühl. Bottom-up Object Detection by Grouping Extreme and Center Points. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019-June:850–859, 1 2019.