STAG: Smart Tools and Applications in Graphics (2024)
A. Caputo, V. Garro, U. Castellani, and A. Giachetti (Editors)

FAST GDRNPP: Improving the Speed of State-of-the-Art 6D Object Pose Estimation

T. Pöllabauer[1,2], A. Pramod[1,3], V. Knauthe[2], M. Wahl[3]

[1]Fraunhofer Institute for Computer Graphics Research, Germany
[2]Technical University Darmstadt, Germany
[3]University of Siegen, Germany

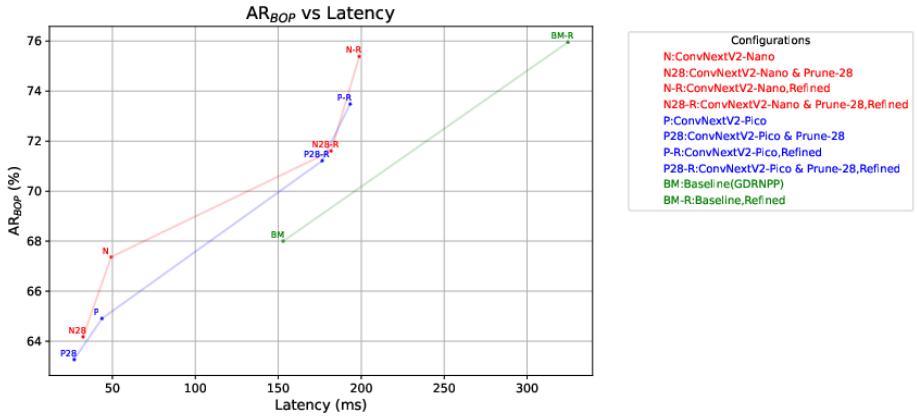


Figure 1: *Average Recall versus inference speed of our approach against the baseline algorithm GDRNPP evaluated and averaged across all 7 BOP classic core datasets. Our fastest model P28 achieves a 82% speed-up with only a 4.7% drop in performance.*

Abstract

6D object pose estimation involves determining the three-dimensional translation and rotation of an object within a scene and relative to a chosen coordinate system. This problem is of particular interest for many practical applications in industrial tasks such as quality control, bin picking, and robotic manipulation, where both speed and accuracy are critical for real-world deployment. Current models, both classical and deep-learning-based, often struggle with the trade-off between accuracy and latency. Our research focuses on enhancing the speed of a prominent state-of-the-art deep learning model, GDRNPP, while keeping its high accuracy. We employ several techniques to reduce the model size and improve inference time. These techniques include using smaller and quicker backbones, pruning unnecessary parameters, and distillation to transfer knowledge from a large, high-performing model to a smaller, more efficient student model. Our findings demonstrate that the proposed configuration maintains accuracy comparable to the state-of-the-art while significantly improving inference time. This advancement could lead to more efficient and practical applications in various industrial scenarios, thereby enhancing the overall applicability of 6D Object Pose Estimation models in real-world settings.

CCS Concepts

• ***Computing methodologies*** → ***Scene understanding;*** *Object detection; Neural networks;*

1. Introduction

Object pose estimation is a fundamental computer vision problem that aims to determine the pose of an object in a given image relative to the camera. Depending on the application needs, the pose can be estimated in varying degrees of freedom (DoF): 3DoF includes only 3D rotation, 6DoF adds 3D translation, and 9DoF further includes the 3D size of the object. 6D object pose estimation, which involves determining both the orientation and translation of

DOI: 10.2312/stag.20241335

an object, is particularly significant in industrial applications. Accurate pose estimation gives an understanding of an object's spatial position, which is vital for tasks in virtual reality, industrial bin-picking, and autonomous driving. Recent state-of-the-art methods for pose estimation are leveraging multi-stage estimation processes and utilize depth information or scene reconstruction for refinement, leading to good results according to the relevant quality metrics, but inference speed is far from real-time. However, for real-world applications inference time, which must align with real-time data acquisition rates (typically 30 frames per second) is a critical quality for many applications.

Therefore our research focuses on reducing inference time to make pose estimation pipelines deployable in real-time settings and/or on low power devices. Using GDRNPP [LZZ*22], an improved version of GDR-Net [WMTJ21], as the baseline due to its high performance and modular design, we propose employing model compression techniques to remove dispensable parameters. This compression aims to accelerate the model and reduce its inference time. We evaluate the impact of different parameter reduction methods and their impact on latency versus accuracy across 7 challenging and relevant datasets, taken from the BOP challenge [HSL*24].

2. Related Work

We will present relevant work on the topic, introducing the BOP challenge and the test datasets (Section 2.1), discussing relevant algorithms (Section 2.2) and details on the selected GDRNPP (Section 2.3), before presenting the ideas of pruning (Section 2.4) and knowledge distillation (Section 2.5).

2.1. BOP Challenge and Datasets

Meaningful evaluation and comparison of algorithms for a given task require unified guidelines, datasets, and metrics. The BOP (Benchmark of Pose Estimation) project, introduced in 2019 [HSD*20] and repeated in 2022 and 2023 [SHL*23, HSL*24], addresses these needs by providing a comprehensive formulation of the 6D pose estimation task, standardized datasets for performance measurement, a set of pose error metrics accounting for different measurement ambiguities and use case requirements, and an online evaluation system with a leaderboard ranking various state-of-the-art methods. Over the years, new tasks have been added, along with updates to the datasets to better evaluate proposed solutions. Our research focuses on the latest 2023 version of the BOP project. We refer to this iteration to define the 6D pose estimation task on seen objects, compare relevant algorithms, and select the most promising as our candidate for further improvement.

BOP provides a selection of relevant datasets, of which 7 are selected as "core classic": LM-O [BKM*14], IC-BIN [HZL*15], YCB-V [XSNF18], T-LESS [HHO*17], ITODD [DUB*17], TUD-L [HMB*18], and HB [KZSI19]. According to the task definition, an algorithm has to be benchmarked on all 7 core classic datasets, which cover a wide range of scenarios including occlusions, cluttered scenes, texture-less objects, varying lighting conditions, and different object types (industrial, household, and toys).

2.2. Algorithm Candidates for Speed up

Table 1: *Results taken from the BOP leaderboard for the 6D localization of seen object tasks [fDOPE]. AR is Average Recall, RGBD uses additional depth input, PBR is using physically-based rendered images for training, Real signals the use of real-world recordings for training, SModel and MModel shows whether a single model per object was trained or whether a single model has to predict multiple objects.*

(a) *Performance of the top 10 models, based on the Average Recall score across all seven core BOP datasets, along with the estimation times in seconds. Among these methods, only the 10th ranked model uses a single-model approach (SModel), RDPN does not specify which approach they use, and the others employ multi-model approaches (MModel).*

Rank	Method	AR	Time
1	GPose2023	85.6	2.670
2	GPose2023-OfficialDetection	85.1	4.575
3	GPose2023-PBR	84.4	2.686
4	GDRNPP-PBRReal-RGBD	83.7	6.263
5	GDRNPP-PBR-RGBD	82.7	6.264
6	ZebraPose-EffnetB4-refined	81.3	2.577
7	GDRNPP-PBRReal-RGBD-Fast	80.5	0.228
8	PFA-Mixpbr-RGBD	80.0	1.193
9	RDPN	79.8	2.429
10	GDRNPP-PBRReal-RGBD-OfficialDet.	79.8	6.406

(b) *Performance of models with fast inference times (measured in seconds) and an Average Recall of at least 60% across the seven core datasets. Among these models, the ones ranked 1st, 11th, and 13th use single-model approaches (SModel), while the others employ multi-model approaches (MModel).*

Rank	Method	AR	Time
1	MRPE-PBRReal-RGB	69.4	0.100
2	GDRNPP-PBRReal-RGBD-MModel-Fast	80.5	0.228
3	GDRNPP-PBRReal-RGB-MModel	72.8	0.229
4	GPose2023-RGB	72.9	0.243
5	ZebraPose-EffnetB4	74.9	0.250
6	ZebraPoseSAT-EffnetB4	72.0	0.250
7	ZebraPoseSAT-EffnetB4 (Detection)	72.0	0.250
8	ZebraPoseSAT-EffnetB4(PBR,Detection)	72.0	0.250
9	GDRNPP-PBR-RGB-MModel	70.2	0.284
10	CosyPose-PRBReal	63.7	0.449
11	GDRNPP-PBRReal-RGB	67.8	0.466
12	ZebraPoseSAT-EffnetB4+ICP	76.5	0.500
13	GDRNPP-PBRReal-RGBD	74.8	0.556

To find a suitable algorithm to accelerate, we take a look at the comprehensive BOP leaderboard. Table 1a shows the top-performing models based on Average Recall (AR) across the seven core datasets of the BOP challenge on the task of localizing seen objects. From top to bottom we find places one to three taken up by GPose, an improved version of GDRNPP. GDRNPP [LZZ*22] is an improved version of GDR-Net [WMTJ21], which again, is an improved version of earlier algorithm CDPN [LWJ19]. Different parameterizations of GDRNPP take up the places 4, 5, 7, and 10. ZebraPose [SSF*22] comes in on place 6, PFA [HFS22] at 8th place, and RDPN [HHC24] is the last entry in the top 10 at 9th

place. Taking inference speed into account and taking another look at the leaderboard sorted by speed we aggregate a second table. Table 1b focuses on especially fast models according to the reported average processing time per image. To create the table, we sort by speed and only select models with an AR score greater than 60 and ignore any algorithm performing worse. We again find all of the above mentioned algorithms in the first positions, with the single additional entry MRPE.

Our final list of candidates, therefore, consists of MRPE, GPose, ZebraPose, PFA, and GDRNPP. Looking through the list we find MRPE and GPose [ZHW*23] unpublished and neither does provide implementation details nor code. Comparing the remaining options, the ZebraPose algorithm [SSF*22] uses binary hierarchical encoding for the vertices of a 3D object. It groups and encodes these vertices, storing the mapping along with the 3D vertices offline. A detector identifies regions of interest in a 2D image, and a fully convolutional neural network (CNN) predicts a multi-layer code. This predicted code is then matched with the stored mapping to produce 2D-3D correspondences. Finally, a Perspective-n-Point (P*n*P) solver is used to estimate the 6D pose. RDPN regresses object coordinates per visible pixel and transforms the coordinates into a residual representation to predict pose. Third candidate PFA [HFS22] estimates an initial pose using a first network and matches this pose with offline-generated templates. The comparison between the retrieved template and the target view in the 2D image is performed by computing the displacement field, which is then transformed into 3D-2D correspondences. The final pose is obtained by solving the P*n*P problem using RANSAC/P*n*P. This method consists of two stages and cannot be trained end-to-end. GDRNPP, to be discussed in detail in Section 2.3, on the other hand is end-to-end differentiable. It follows a modular design, which allows to improve each part of the estimation pipeline separately. At the same time it is the fastest of the top 10 algorithms (7^{th} in Table 1a) and 2^{nd} in Table 1b, and the base algorithm for best performing algorithm GPose, which, together with other previous extensions [PEKK24,PLK*24] indicate a flexible algorithm and a worthwhile target for acceleration. ZebraPose and RDPN, in contrast, are noticeably less accurate (5^{th}-8^{th} place in Table 1b) or twice (12^{th} place in Table 1b) / ten-times slower (RDPN on 9^{th} place in Table 1a).

Based on this comparison, we choose GDRNPP as our base algorithm. Among the variants of GDRNPP, we choose to select the single-model per dataset version as a baseline. Considering that our intent is to reduce run-time by scaling down the number of parameters, this way we will instantly see, when we reached the lower bound on the necessary complexity to reach a certain level of performance. We also argue that, by doing so, improvements to the single-model per dataset should propagate to the multi-model approach, where the whole pipeline only needs to fit a single object. We discuss the details of GDRNPP next.

2.3. GDRNPP

GDRNPP is an enhanced version of GDR-Net [WMTJ21] illustrated in Figure 2, using deep neural networks to directly regress required features. It consists of three sub-modules: Object detection, feature prediction, and pose estimation. Also, there is a 4^{th}

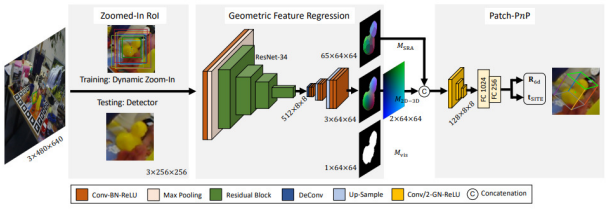


Figure 2: *Processing pipeline of GDR-Net as presented in [WMTJ21]. GDRNPP follows an identical setup, only swapping ResNet with ConvNext, and predicting an additional, amodal object mask. Given an RGB image, the model takes a zoomed-in Region of Interest (RoI) of the image as input and extracts features using the backbone (1^{st} module). Based on the encoding, a geometric decoder head then predicts intermediate geometric feature maps (2^{nd} module) M_{SRA}, M_{vis}, M_{2D-3D}. These features are fed into the Patch-PnP module (3^{rd} module) to regress the rotation and translation.*

optional step for pose refinement. To guarantee a fair comparison, we will use the official BOP detections and will not concern ourselves with the detection step at all. The pose refinement module includes two options: Coupled Iterative Refinement [LTGD22] and a fast refinement setup. For our purposes, we will use the fast refinement setup to evaluate the speed improvements introduced by our method. Our primary focus, however, is on both the feature prediction and the pose estimation modules (2^{nd} and 3^{rd} module).

A forward pass through GDR-Net works as follows (and is functional-wise the same for GDRNPP): An RGB image is fed to the detector, which returns a set of detections. Next, GDR-Net applies dynamic zoom in [LWJ19], to create a range of different object crops, making the algorithm more robust to variances in detection bounding box predictions. The object crops are resized to 256×256 pixels during the training phase. During testing, it uses inputs from the detector (we use the official YOLOX [GLW*21] detections). These inputs are then fed into the geometric feature module, a ResNet-34 [HZRS16] backbone, which extracts image feature maps of resolution $512 \times 8 \times 8$ from the zoomed-in RoI, and a decoder head, which subsequently extracts intermediate geometric features from the image feature maps, namely surface region attention maps M_{SRA}, visible/modal object mask M_{vis}, and 2D-3D correspondence maps M_{2D-3D}. Finally, a Patch P*n*P module directly regresses the rotation (3DoF) and translationb(3DoF) from the learned features M_{SRA} and M_{2D-3D}. The main architectural differences between GDR-Net and GDRNPP are the use of ConvNext-B [LMW*22] as a more advanced backbone for feature extraction, the addition of amodal mask prediction M_{amo}, in addition to modal mask M_{vis}, as well as more involved image augmentation.

2.4. Pruning

The uses and effects of pruning and quantization in deep learning are well documented [LGW*21, CWZZ17]. Pruning is the process of reducing the number of learnable parameters in a network by removing those that only negligibly add to the function approxima-

tion. Pruning might help to prevent overfitting and improve speed. Quantization is the process of reducing the expressiveness of parameters by switching to lower bit widths, such as 8-bit integers and even smaller. With techniques such as Nvidia's automatic mixed precision [NVI24] built right into TensorFlow and Pytorch, we focus on pruning. Pruning involves two main steps: first, selecting and removing parameters, and second, retraining the pruned model for a small number of epochs to regain performance, called fine-tuning. Pruning can be done in one-shot, where the network is pruned to the desired degree and then fine-tuned, or iteratively, where the model is partially pruned and retrained multiple times. A further distinction is made between structured [LKD*22, HX23] and unstructured [BGOFG20] pruning. Structured pruning removes entire channels or layers, while unstructured pruning removes individual weights. [VA22] categorizes common pruning techniques into, first, magnitude based, second, clustering based, and third, sensitivity analysis based methods.

2.5. Knowledge Distillation

Knowledge distillation (KD) in deep learning describes the process of extracting knowledge from one model (called the teacher) and transferring it to another model (the so called student) [CWZZ17, GYMT21]. The knowledge transfer can occur from the last layer, the entire teacher model, or specific parts of it, depending on the method used. KD can be applied to arbitrary domains of deep learning, but has been especially interesting for computer vision because of the large networks usually found within the domain. Wang and Yoon [WY21] provide an in-detail survey on the student-teacher framework applied to computer vision. Also, a very useful property of knowledge distillation is the fact that models tend to learn faster from a teacher, than from ground truth data [PL19].

3. Methodology

Next we discuss our approaches to reduce the run-time of GDRNPP. We start introducing a selection of relevant alternative backbones (Section 3.1), discuss the possibilities of shrinking the decoder head by reducing the number of predicted features (Section 3.2), applying pruning (Section 3.3), as well as knowledge distillation (Section 3.4).

3.1. Backbone Selection

GDRNPP originally uses a ConvNext-B model [LMW*22] as backbone, featuring 89 million learnable parameters, a top-1 accuracy of 83.8% on the ImageNet-1K dataset [DDS*09], and 15.6 GFLOPS. For context, the original GDR-Net used ResNet-34 [HZRS16] achieved a top-1 accuracy of 73.31% with only 3.66 GFLOPS. It is fair to say that ConvNext-B demonstrates better feature extraction capabilities than ResNet-34 and, as a result, much of GDRNPPs outperformance can be credited to the more expressive backbone. The increase in performance between GDR-Net and GDRNPP due to the change in backbone indicates the backbone being a crucial factor for performance. Our goal therefore becomes to find a backbone that performs similar to ConvNext-B, all the while being noticeably faster.

Table 2 lists a selection of, in our eyes, interesting backbone choices, as well as the original ConvNext-B. The data for the first seven models (ConvNext, FasterVIT, and EfficientNet-V2) come from the research by [HHY*24], measured on A100 GPUs with a batch size of 128. The ConvNext-V2 models [WDH*23] are evaluated on RTX 3090 GPUs with a batch size of 256. The EfficientNet-V2-B0 is benchmarked on a V100 GPU with a batch size of 128. We based our selection on 4 criteria: total number of learnable parameters, GFLOPS, image throughput, and top-1 classification performance on ImageNet. Since the different models are not benchmarked on the same, though similarly powerful hardware, these numbers need to be taken with a grain of salt, though should give a good estimate for comparisons. To find the best candidates for our pipeline, we will test all of the listed backbones integrated in GDRNPP.

Model	Params	GFLOPs	Throughput	Top-1
ConvNext-B	88.6	15.4	1485	83.8
ConvNext-S	50.2	8.7	2008	83.1
FasterVIT-0	31.4	3.3	5802	82.1
FasterVIT-1	53.4	5.3	4188	83.2
FasterVIT-2	75.9	8.7	3161	84.2
FasterVIT-3	159.5	18.2	1780	84.9
ConvNext-V2-T	28.64	4.47	1452.72	83.8
ConvNext-V2-N	15.6	2.45	2300.18	82.1
ConvNext-V2-P	9.1	1.37	3274	80.3
EfficientNet-V2-B0	7	0.5	5739	78.7
EfficientNet-V2-S	21.5	8.0	1735	83.9

Table 2: *Backbone candidates. We identify the following relevant criteria for our selection: number of parameters (millions), GFLOPs, throughput (images per second), and top-1 accuracy (%).*

3.2. Region Ablation

In the paper presenting GDR-Net, the authors test how the number of predicted regions for the attention maps M_{SRA} influence overall accuracy evaluated on LM. They find that already without M_{SRA} (0 regions) performance is good, while increasing the number of regions leads to slight performance gains. This behavior saturates at around 64 regions, which is the final value they settled with, arguing that this is a good trade-off between accuracy and memory requirements. We on the other hand are most interested in the effect on speed, wherefore we re-run the test to answer the question, how different numbers of regions influence run-time.

3.3. Pruning

We apply structured pruning with an L_1-norm based filter pruning method [LKD*22] to the estimation pipeline, removing weights with smaller magnitudes, assuming they have a negligible effect on the model's output. The pruning procedure works as follows: We select the weights of filters in a specific 2D convolutional layer and calculate the L_1 norm along the channel dimension. Filters are then ranked in ascending order of their norms, and the lowest D-ranking filters are removed. Consequently, the output channels from these removed filters and the kernels applied to these channels are also removed, as illustrated in Figure 3. Given an input feature map

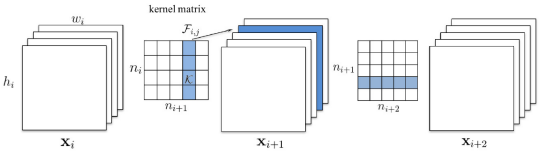


Figure 3: *Illustration of L_1-norm filter pruning as found in [LKD*22]. $F_{i,j}$ is the filter activation on layer x_i. Removal of a filter directly propagates through the network, leading to additional filters being removed further down.*

x_i with shape $w_i \times h_i \times n_i$ and a convolutional layer with parameters F_i having shape $n_{i+1} \times n_i \times k \times k$, the total operations are $n_{i+1} \times n_i \times k^2 \times h_i \times w_i$. Pruning a kernel $F_{i,j}$ removes its corresponding feature map, reducing operations by $n_i \times k^2 \times h_i \times w_i$. This also affects the next convolutional layer F_{i+1}, further reducing operations by $n_{i+2} \times k^2 \times h_{i+2} \times w_{i+2}$. We apply pruning to both the geometric head, as well as the Patch P*n*P module. However, before starting the pruning, the group normalization layers need to be put into consideration, because stochastically selecting and removing layers could disrupt group normalization. Therefore, we remove groups of eight and four filters per parameter D for each module respectively, ensuring the grouping remains intact. Figures 4 and 5 show the parameterized representation of the layer architecture for both modules with respect to the hyperparameter D, controlling the amount of pruning with a higher value leading to more aggressive pruning.

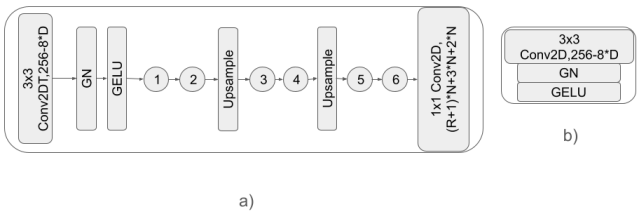


Figure 4: *Illustration of L_1 norm filter pruning applied to the geometric head. The layer architecture is adjusted by $8 \times D$.*

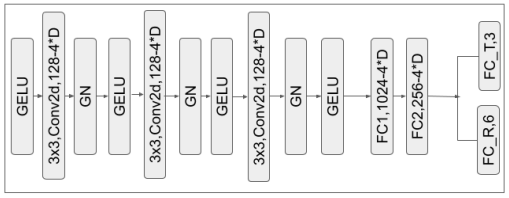


Figure 5: *Illustration of L_1 norm filter pruning applied to the Patch PnP module. The layer architecture is adjusted by $4 \times D$.*

Once the pruned models are extracted from the larger model following the process detailed above, they are fine-tuned for a few epochs until their performance is close to that of the original, unpruned model.

3.4. Distillation

As with any learning problem, the choice of loss function is also very important for knowledge distillation. A typical loss function between the student and teacher is the Kullback-Leibler (KL) divergence loss [HVD15], as shown in Equation 1. This loss is calculated between the softened probability distributions of the teacher and student predictions given in Equation 2, controlled by the hyper-parameter τ.

$$L_{KL}(p_s(\tau), p_t(\tau)) = \tau \sum_j p_{t,j}(\tau) \log \frac{p_{t,j}(\tau)}{p_{s,j}(\tau)} \tag{1}$$

$$p_{f,k}(\tau) = \frac{\exp(z_{f,k}/\tau)}{\sum_{j=1}^{K} \exp(z_{f,j}/\tau)} \tag{2}$$

Kim et al. [KOK*21] compare the effectiveness of KL loss and mean squared error (MSE) loss, finding that MSE performs better for regression-based models. Since our solution space involves regression, we chose MSE as the distillation loss, as shown in Equation 3, where N is the length of the logits vector, $z_{s,i}$ is the student logits, and $z_{t,i}$ is the teacher logits.

$$\text{MSE} = \frac{1}{N} \sum_{i=1}^{N} (z_{s,i} - z_{t,i})^2 \tag{3}$$

For our distillation process, we select the pre-trained GDRNPP model, specific to each dataset, as the teacher model. Student models are chosen based on the performance of selected backbones, as listed in Section 3.1 and evaluated in Section 4.3. Given that the output channel dimensions of the student model backbones differ from those of the teacher model, directly applying a loss function without aligning feature dimensions would be problematic. To address this, we initially up-sample the output of the student model's backbone to match the feature dimensions of the teacher model's backbone using a $1x1$ convolutional layer. We then normalize the outputs of both the student and teacher models and apply the MSE loss, as described above. The computed loss is solely used to supervise the student backbone. The geometric head and Patch-P*n*P module are supervised based on their respective loss functions as found in the original GDRNPP [LZZ*22].

4. Evaluation

We introduce the task and relevant evaluation metrics (Section 4.1), describe the implementation details (Section 4.2), evaluate the selected backbones integrated in GDRNPP (Section 4.3), and investigate the influence of reducing the number of regions in M_{SRA} (Section 4.4), the influence of pruning both the geometric head, as well as the Patch P*n*P module (Section 4.5), and the influence of knowledge distillation (Section 4.6). Finally we conclude with combining our learnings into our final approach in Section 4.7.

4.1. Task Description and Metrics

For a fair comparison we follow the task and evaluation description of the BOP challenge for the task of localization of seen objects. Seen objects refers to the algorithm being allowed to see a set of images of the objects in question during training. The training and test sets of images are disjunct. We evaluate the following metrics: latency of the model, Average Recall following the 3 metrics Visible Surface Discrepancy (VSD), Maximum Symmetry-Aware Surface Distance (MSSD), Maximum Symmetry-Aware Projection Distance (MSPD) as defined by BOP, as well as the AR of Average Distance (with Symmetry) (ADD/ADD-S) metric, which is another common metric in the field. The metrics are defined as follows:

Latency. We measure latency as the forward pass time over all test images per dataset divided by the number of images in the dataset. All measurements, including those for the geometric head and Patch-P*n*P, are taken on the same GPU using the standard time module's performance counter, reported in milliseconds.

Average Recall. Recall is the ratio of truly positive predictions relative to all positive predictions as defined in Equation 4.

$$\text{Recall} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}} \tag{4}$$

AR defines thresholds for a given metric. A result is considered correct, if the prediction falls within the defined threshold. Recall scores are calculated for multiple thresholds and then averaged. For BOP the three recall scores for VSD, MSSD, and MSPD are calculated and averaged to get the final AR_{BOP} score.

VSD. Equation 5 calculates VSD by comparing Euclidean distances between distance maps $(\hat{D}, \bar{D})$ rendered from estimated and ground truth poses. The average number of pixels violating the distance threshold within the union of predicted and ground truth masks gives e_{VSD}, with τ varying from 5% to 50% of the object diameter and correctness thresholds ranging from 0.05 to 0.5.

$$e_{\text{VSD}}(\hat{D}, \bar{D}, \hat{V}, \bar{V}, \tau) = \text{avg}_{p \in \hat{V} \cup \bar{V}} \begin{cases} 0 & \text{if } p \in \hat{V} \cap \bar{V}, \left|\hat{D}(p) - \bar{D}(p)\right| < \tau \\ 1 & \text{otherwise} \end{cases} \tag{5}$$

MSSD and MSPD.

$$e_{\text{MSSD}} = \min_{\mathbf{S} \in \mathcal{S}} \max_{\mathbf{x} \in \mathcal{M}} \left\| (\hat{P}\mathbf{x}) - (\bar{P}\mathbf{S}\mathbf{x}) \right\| \tag{6}$$

$$e_{\text{MSPD}} = \min_{\mathbf{S} \in \mathcal{S}} \max_{\mathbf{x} \in \mathcal{M}} \left\| proj(\hat{P}\mathbf{x}) - proj(\bar{P}\mathbf{S}\mathbf{x}) \right\| \tag{7}$$

Equations 6 and 7 define MSSD and MSPD respectively. Each vertex in the 3D model M is transformed using the ground truth $\bar{P}$ and estimated pose $\hat{P}$, and the euclidean distances between the transformed 3D points and their 2D projections are computed separately. S is a set of global symmetry transformations. The minimum distances across all symmetries, e_{MSSD} and e_{MSPD}, are calculated, with e_{MSSD} using the same threshold τ and e_{MSPD} using a threshold of $5r$ to $50r$ in $5r$ steps, where $r = w/640$ and w is the image width.

ADD(-S).

$$e_{\text{ADD}} = \frac{1}{m} \sum_{\mathbf{x} \in \mathcal{M}} \left\| (\hat{P}\mathbf{x}) - (\bar{P}\mathbf{x}) \right\| \tag{8}$$

For the ADD metric, the average distance between model points transformed using the estimated and ground truth poses is calculated using Equation 8. Recall is computed for correctness thresholds of 2%, 5%, and 10% of the object's diameter and averaged, which can be made symmetry-aware (ADD-S) by measuring the distance to the closest point on the ground truth model.

4.2. Implementation Details

Our experiments are conducted using a HPC cluster equipped with NVIDIA A100 SXM4 40GB GPUs. We use the BOP toolkit for evaluation, but measure latency ourselves due to instability in the toolkit's latency measurements. Among the 7 core datasets, we select LM-O for our ablation studies because it has fewer training instances but presents challenging scenarios, making it suitable for both computational and benchmarking purposes. We test all latency improvement methods on this dataset to inform our decision process on what to keep for our final model. Based on the performance improvements demonstrated by each method, we generalize to select the configuration that offers the best balance between speed and accuracy discussed in Section 4.7. As for data, we follow the approach of using PBR and REAL data as provided by the BOP challenge.

4.3. Backbone Selection

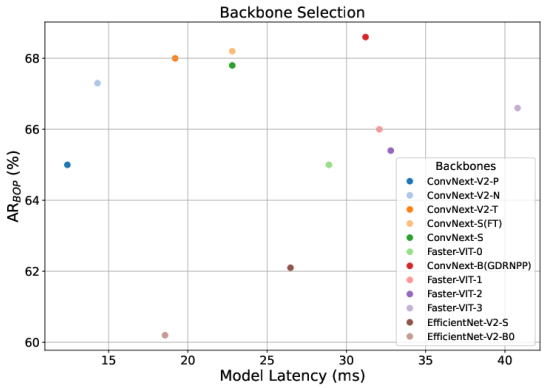


Figure 6: *Performance of various backbones when integrated into GDRNPP.*

The backbones listed in Table 2 are used to estimate object poses in the LM-O dataset. All models are trained for 40 epochs with the learning rate and hyper-parameters as reported for GDRNPP. The trained models are tested on the official BOP test data, which comprises 200 images with 1445 object instances across various scenes, all eight objects being visible in almost every image. Figure 6 shows AR results for different GDRNPP backbone variants.

Despite FasterVIT variants having a much higher frame rate than

ConvNext-B in ImageNet classification, their performance in pose estimation varies. The fastest variant, FasterVIT-0, is 7.39% faster than GDRNPP but suffers a 5.3% drop in AR. Other FasterVIT variants are slower and perform below the benchmark. EfficientNet variants show improved speed over ConvNext-B: EfficientNet-V2-B0 is 40.53% faster, and EfficientNet-V2-S is 15.17% faster. However, their performance drops by 12.24% and 9.47%, respectively. ConvNext variants demonstrate promising performance, especially ConvNext-V2-P: The maximum recall score drop is a minor 5.24% while latency is reduced by 60%.

As a side note we want to highlight the performance discrepancy between backbone performance on classification and the performance we see in our pose estimation pipeline. There is a wide range of potential reasons for that behavior, such as task complexity, task specific feature requirements and model suitability.

Selection Criteria. To achieve an inference rate close to 30 frames per second or above, we have a budget of approximately 33.33 ms per image. With GDRNPP requiring an off-the-shelf detector like YOLOX-X (we measured an inference time of 8 ms), we have 25.3 ms left for pose estimation. Therefore, we focus on backbones with latencies below 25 ms, narrowing our selection to ConvNext-V2-T, ConvNext-V2-N, and ConvNext-V2-P models. ConvNext-V2-N reduces latency by about 5 ms compared to ConvNext-V2-T, with only a 1.02% difference in AR scores. Given this trade-off, ConvNext-V2-N is chosen for its balanced performance and ConvNext-V2-P for its very low latency, allowing an extra 7-8 ms that might be invested in the optional pose refinement step. Also, keep in mind that inference time increases with the number of objects within a scene, so larger sets tend to require substantially more time.

4.4. Number of Regions

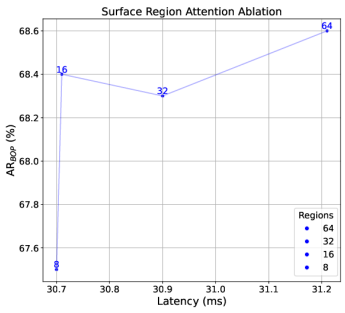
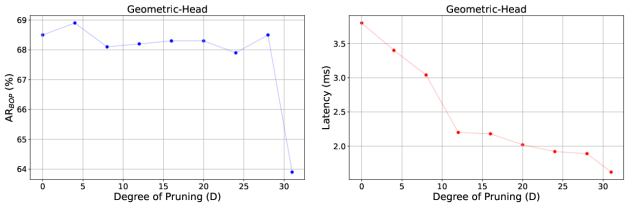


Figure 7: *Illustration of performance of GDRNPP with varying number of predicted regions.*

As described in Section 3.2, we evaluate the influence of the number of regions used with surface attention on run-time. The original GDRNPP achieves an AR of 68.6% when using 64 regions, with an inference time of 31.2ms. We test different numbers of regions and plot the results in Figure 7. As expected based on the GDR-Net paper, performance drops with lower number of regions (1.6% with only 8 regions). 16, 32, and 64 regions offer similar AR scores, while inference speed varies by 0.5ms between the choice of 16 and 64 regions. These numbers indicate that we can only expect limited gains, when decreasing the number of regions. Considering the ablation results of original GDR-Net, which show 64 regions to be the sweet spot, we decide not to alter from their choice and also use 64 regions with our faster versions.

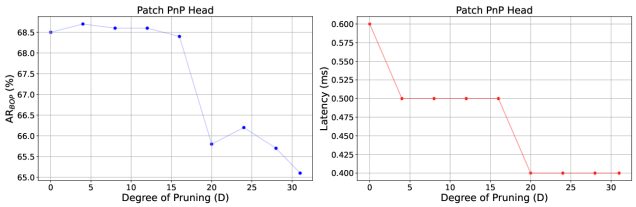
4.5. Pruning

Having selected a faster backbone in Section 4.3, we now focus on the geometric head and the Patch P*n*P module. The experiments described in Section 3.3 yield the following observations: as expected and visualized in Figure 8, increasing the degree of pruning improves latency. The original model's geometric head has a latency of 3.8ms. Initial pruning improves latency by approximately 10%. For higher degrees of pruning, latency improves to about 1.89ms at D=28. The maximum degree of pruning in our setup is D=31, reducing latency by 57.36%. Interestingly, AR remains stable until D=28 as long as we fine-tune the pruned model. At the maximum pruning degree, the AR drops by only 6.7%, which is a small loss compared to the reduction in latency.



(a) *(b)*

Figure 8: *(a) Performance of the geometric head with increasing level of pruning. (b) Latency while increasing the level of pruning.*



(a) *(b)*

Figure 9: *(a) Performance of Patch PnP with increasing level of pruning. (b) Latency while increasing the level of pruning.*

Turning our attention to the the Patch P*n*P module, as illustrated in Figure 9, the unpruned Patch P*n*P module has 15.6% lower latency than the geometric head due to fewer operations, indicating that the pruning of the Patch P*n*P module is less significant and more attention should be spent on the geometric head. Also, at higher pruning degrees (D=24, 28, 31), latency stagnates around 0.4ms. Performance drops by 4-5% only after pruning exceeds D=16, without significant latency improvement. Based on these observations, we decide to set our pruning hyperparameter for the geometric head to D=28 and for the Patch P*n*P to D=16, a choice that leads to no loss in performance, all the while improving latency.

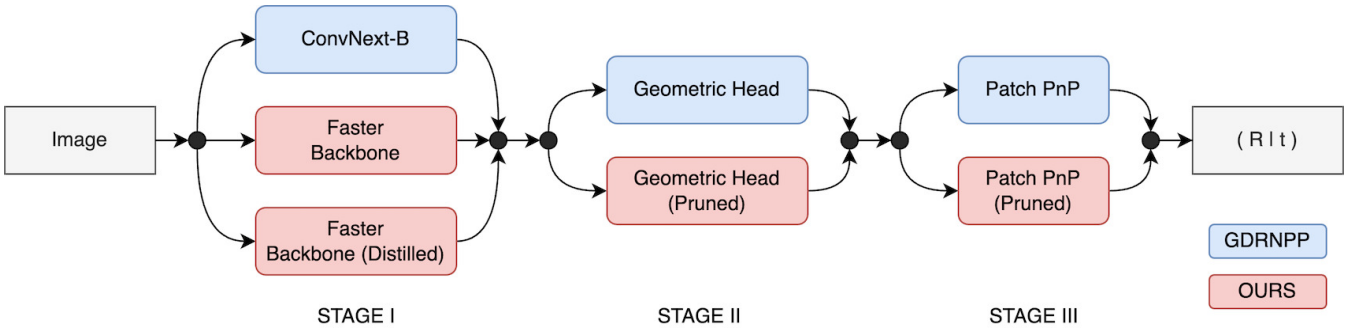


Figure 10: *Overview of our acceleration options in comparison to the baseline algorithm.*

Metrics	N	N28	P	P28	N-R	N28-R	P-R	P28-R	BM	BM-R
AR (%)	67.37	64.18	64.91	63.27	75.38	71.6	73.48	71.21	68.0	75.9
Latency (ms)	49.2	32.2	43.79	26.8	198.8	181.8	193.2	176.4	152.9	324.6

Table 3: *Scores and corresponding measured latency aggregated across the seven core datasets. P28 is the fastest configuration, while N-R is the best performing.*

Backbone	AR wo. Dist.	AR w. Dist.
ConvNext-B	68.6	-
ConvNext-S	68.2	69.0
ConvNext-V2-T	68.0	68.7
ConvNext-V2-N	67.3	68.0
ConvNext-V2-P	65.0	66.0

Table 4: *Comparison of distillation performance using 4 relevant bacbkones. We see a slight outperformance when applying distillation across all experiments.*

4.6. Distillation

We conduct the distillation process as described in Section 4.6 and compare the performance of the distilled student models to their non-distilled counterparts. The results from the experiment using different student backbones distilled from the GDRNPP model on the LM-O dataset are listed in Table 4. The four fastest backbones from Section 4.3 were chosen as the student backbones for this experiment. The original GDRNPP teacher model (ConvNext-B) achieves an Average Recall of 0.686. We see that every distilled model (w. Dist.) outperforms the model, which is trained directly on the data (wo. Dist) and it is evident that distillation helps improve the model's score. For ConvNext-S and ConvNext-V2-T, the models perform on par with the original ConvNext-B model even without distillation. However, distillation helps these models match or slightly exceed the original model's performance. Overall distillation adds a slight performance increase to the smaller backbones, at the cost of requiring a trained base model.

4.7. Full Approach

Finally, we combine our learnings to create models for different run-time requirements. Figure 10 shows where different optimization techniques fit within the pipeline. We present two default configurations: The first opts to maximize inference speed and based on ConvNext-Pico (configuration P), the second tries to balance speed and the loss in accuracy and is based on ConvNext-Nano (configuration N).

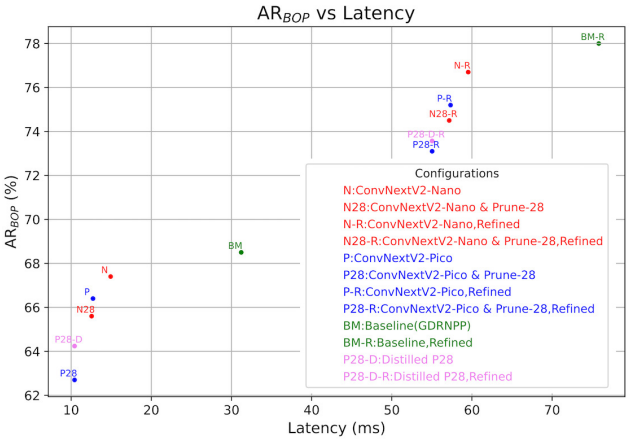


Figure 11: *Performance of our selected configurations, using distillation and/or pruning on the LM-O dataset.*

Based on our extensive analysis we have found the following methods to be the most promising: First, selection of a faster backbone, second backbone distillation, third pruning the geometric head and the Patch P*n*P module. We combine these into the following set of configurations: Models N and P are only replacing the backbone. 28 indicates that we use a pruned geometric head. -D shows that we use a distilled backbone. For all configurations, we additionally test the influence of fast depth refinement, indicated by -R. For comparison with the baseline, we add BM and BM-R representing the default configuration of GDRNPP. Results of the methods on LM-O are presented in Figure 11. Based on these re-

sults, we see that all models fall at different spots in the AR vs. latency ratio, giving practitioners the chance to choose what best fits their needs.

For a more complete picture of how the methods perform on a wide range of different objects, we continue to evaluate our configurations on the remaining 6 BOP core datasets, with the omission of backbone distillation, since we found it impractical due to the training overhead. Distillation necessitates the training of an expressive teacher model, dramatically increasing overall training time and complexity. Based on the results on LM-O, we see a noticeable improvement with distillation and depending on the use case, the additional training is acceptable. However, for our remaining evaluation, we focus on faster backbones and pruning.

Figure 1 shows the average performance of our selected configurations N and P, as well as the baseline BM, with and without pruning and with and without refinement. The baseline model BM has an AR score of 68%, which improves to 75.9% with refinement, while the latency increases from 153ms to 324.6ms. The unpruned, unrefined variants (N and P models) show a latency improvement of 65% and 75% compared to the baseline, with a negligible performance drop for N and a 3% drop for P. Pruning further improves latency, with N28 showing a 79% improvement and P28 an 82% improvement, accompanied by a 3.8% and 4.7% performance drop, respectively. Refinement improves scores for all configurations. Compared to the baseline BM-R, the unpruned models N-R and P-R have a minimal recall drop of 1.6% and 2.4%, respectively, with a speed improvement of 38.7% and 40%. Pruned versions are even faster, with N28-R and P28-R showing a recall drop of about 5% and a latency improvement of about 44% compared to BM-R. The fastest configuration is P28, with a 12.6% score drop compared to BM-R, but a 92% improvement in latency. Our best-performing configuration is N-R, with less than a 1% score drop and a 38.7% latency improvement. All results are shown in Table 3.

4.7.1. Parameter Considerations

Metrics	N	N28	P	P28	BM
# Parameters (M)	29.4	24.298	22.68	17.83	102.87

Table 5: *Number of parameters in millions for various configurations on the YCB-V dataset.*

Table 5 shows the total number of learnable parameters for each configuration (without refinement) on the YCB-V dataset. The baseline model has 102.87M parameters. Switching to ConvNext-V2-Nano (N) and ConvNext-V2-Pico (P) reduces parameters by 71% and 78%, respectively. Pruning the geometric head by D=28 reduces approximately 5M parameters for both models. The pruned models N28 and P28 have a parameter reduction of 78.5% and 82.6%, respectively. Similar trends are observed across all datasets, with minor variations depending on the number of objects.

4.7.2. Real-time Considerations

As stated above, a typical camera framerate is 30 fps, giving a total of 33.33ms of processing time per frame. Looking at our results averaged across the seven datasets, all of different complexity, we see that only P28 and N28 stay below that budget. To illustrate the effect of different datasets, we report the results of both configurations per dataset in Table 6. We see vast differences in inference speed per dataset. Larger datasets such as T-LESS, ITODD, HB, and ICBIN tend to have more objects per image and much higher compute requirements. At the same time, our configurations move much closer towards real-time across all datasets, fullfilling the 30 fps target on LM-O, YCB-V, ITODD, and TUD-L. Our fastest configuration achieves 16.6 fps on the slowest dataset (ICBIN) compared to 2.35 fps with the original algorithm. As stated above, all of these numbers are benchmarked on a single A100 GPU. Bringing this level of real-time performance to devices with lower compute capabilities will need further reductions in computational requirements.

		LM-O	ICBIN	YCB-V	T-LESS	ITODD	TUD-L	HB
P28	AR (%)	62.7	59.1	68.4	68.4	29	78.5	76.8
	Lat (ms)	10.4	60.1	9.8	35.3	22.7	8.6	41.1
N28	AR (%)	65.6	59.9	74.2	68.9	25.4	80.6	74.7
	Lat (ms)	12.6	76	10.8	37.4	27.9	9.5	51.5
BM	AR (%)	68.5	63.4	76.8	77.6	26	82.8	80.9
	Lat (ms)	31.2	425.6	27.7	167.3	121.7	13	284.2

Table 6: *AR score and inference time of our fastest configurations and the baseline reported per dataset.*

5. Conclusions and Future Work

We explored acceleration techniques for the 6D pose estimator GDRNPP. Surface region attention improved performance but not inference time, while smaller backbones enhanced speed with minimal performance loss. Knowledge distillation offered slight gains but requires the training of 2 individual models. The geometric head has a greater impact on inference time than the Patch P*n*P modules, leading us to reduce its parameters via pruning. We developed configurations that balance accuracy and speed, notably configuration P28, which has a 4.7% performance drop and an 82% latency improvement. Our findings move state-of-the-art 6D pose estimation a step closer towards real-time inference.

References

[BGOFG20] BLALOCK D., GONZALEZ ORTIZ J. J., FRANKLE J., GUTTAG J.: What is the state of neural network pruning? *Proceedings of machine learning and systems 2* (2020), 129–146. 4

[BKM*14] BRACHMANN E., KRULL A., MICHEL F., GUMHOLD S., SHOTTON J., ROTHER C.: Learning 6d object pose estimation using 3d object coordinates. In *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part II 13* (2014), Springer, pp. 536–551. 2

[CWZZ17] CHENG Y., WANG D., ZHOU P., ZHANG T.: A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282* (2017). 3, 4

[DDS*09] DENG J., DONG W., SOCHER R., LI L.-J., LI K., FEI-FEI L.: ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09* (2009). 4

[DUB*17] DROST B., ULRICH M., BERGMANN P., HÄRTINGER P., STEGER C.: Introducing mvtec itodd — a dataset for 3d object recognition in industry. In *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)* (2017), pp. 2200–2208. doi:10.1109/ICCVW.2017.257. 2

[fDOPE] FOR 6D OBJECT POSE ESTIMATION B.: Leaderboards: Model-based 6D localization of seen objects – BOP-Classic-Core. https://bop.felk.cvut.cz/leaderboards/pose-estimation-bop19/bop-classic-core/. 2

[GLW*21] GE Z., LIU S., WANG F., LI Z., SUN J.: YOLOX: exceeding YOLO series in 2021. *CoRR abs/2107.08430* (2021). URL: https://arxiv.org/abs/2107.08430, arXiv:2107.08430. 3

[GYMT21] GOU J., YU B., MAYBANK S. J., TAO D.: Knowledge distillation: A survey. *International Journal of Computer Vision 129*, 6 (2021), 1789–1819. 4

[HFS22] HU Y., FUA P., SALZMANN M.: Perspective flow aggregation for data-limited 6d object pose estimation. In *European Conference on Computer Vision* (2022), Springer, pp. 89–106. 2, 3

[HHC24] HONG Z.-W., HUNG Y.-Y., CHEN C.-S.: Rdpn6d: Residual-based dense point-wise network for 6dof object pose estimation based on rgb-d images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops* (June 2024), pp. 5251–5260. 2

[HHO*17] HODAŇ T., HALUZA P., OBDRŽÁLEK Š., MATAS J., LOURAKIS M., ZABULIS X.: T-LESS: An RGB-D dataset for 6D pose estimation of texture-less objects. *IEEE Winter Conference on Applications of Computer Vision (WACV)* (2017). 2

[HHY*24] HATAMIZADEH A., HEINRICH G., YIN H., TAO A., ALVAREZ J. M., KAUTZ J., MOLCHANOV P.: Fastervit: Fast vision transformers with hierarchical attention. In *The Twelfth International Conference on Learning Representations* (2024). 4

[HMB*18] HODAŇ T., MICHEL F., BRACHMANN E., KEHL W., GLENT BUCH A., KRAFT D., DROST B., VIDAL J., IHRKE S., ZABULIS X., SAHIN C., MANHARDT F., TOMBARI F., KIM T.-K., MATAS J., ROTHER C.: BOP: Benchmark for 6D object pose estimation. *European Conference on Computer Vision (ECCV)* (2018). 2

[HSD*20] HODAŇ T., SUNDERMEYER M., DROST B., LABBÉ Y., BRACHMANN E., MICHEL F., ROTHER C., MATAS J.: Bop challenge 2020 on 6d object localization. In *Computer Vision–ECCV 2020 Workshops: Glasgow, UK, August 23–28, 2020, Proceedings, Part II 16* (2020), Springer, pp. 577–594. 2

[HSL*24] HODAN T., SUNDERMEYER M., LABBE Y., NGUYEN V. N., WANG G., BRACHMANN E., DROST B., LEPETIT V., ROTHER C., MATAS J.: Bop challenge 2023 on detection, segmentation and pose estimation of seen and unseen rigid objects. *arXiv preprint arXiv:2403.09799* (2024). 2

[HVD15] HINTON G., VINYALS O., DEAN J.: Distilling the knowledge in a neural network. *stat 1050* (2015), 9. 5

[HX23] HE Y., XIAO L.: Structured pruning for deep convolutional neural networks: A survey. *IEEE transactions on pattern analysis and machine intelligence* (2023). 4

[HZL*15] HODAN T., ZABULIS X., LOURAKIS M., OBDRZÁLEK S., MATAS J.: Detection and fine 3d pose estimation of texture-less objects in rgb-d images. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2015, Hamburg, Germany, September 28 - October 2, 2015* (2015), pp. 4421–4428. doi:10.1109/IROS.2015.7354005. 2

[HZRS16] HE K., ZHANG X., REN S., SUN J.: Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (2016), pp. 770–778. 3, 4

[KOK*21] KIM T., OH J., KIM N., CHO S., YUN S.: Comparing kullback-leibler divergence and mean squared error loss in knowledge distillation. *CoRR abs/2105.08919* (2021). URL: https://arxiv.org/abs/2105.08919, arXiv:2105.08919. 5

[KZSI19] KASKMAN R., ZAKHAROV S., SHUGUROV I., ILIC S.: Homebreweddb: Rgb-d dataset for 6d pose estimation of 3d objects. *International Conference on Computer Vision (ICCV) Workshops* (2019). 2

[LGW*21] LIANG T., GLOSSNER J., WANG L., SHI S., ZHANG X.: Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing 461* (2021), 370–403. 3

[LKD*22] LI H., KADAV A., DURDANOVIC I., SAMET H., GRAF H. P.: Pruning filters for efficient convnets. In *International Conference on Learning Representations* (2022). 4, 5

[LMW*22] LIU Z., MAO H., WU C.-Y., FEICHTENHOFER C., DARRELL T., XIE S.: A convnet for the 2020s. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (2022), pp. 11976–11986. 3, 4

[LTGD22] LIPSON L., TEED Z., GOYAL A., DENG J.: Coupled iterative refinement for 6d multi-object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 6728–6737. 3

[LWJ19] LI Z., WANG G., JI X.: Cdpn: Coordinates-based disentangled pose network for real-time rgb-based 6-dof object pose estimation. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)* (October 2019). 2, 3

[LZZ*22] LIU X., ZHANG R., ZHANG C., FU B., TANG J., LIANG X., TANG J., CHENG X., ZHANG Y., WANG G., JI X.: Gdrnpp. https://github.com/shanice-l/gdrnpp_bop2022, 2022. 2, 5

[NVI24] NVIDIA: Automatic mixed precision for deep learning. https://developer.nvidia.com/automatic-mixed-precision, 2024. 4

[PEKK24] PÖLLABAUER T., EMRICH J., KNAUTHE V., KUIJPER A.: Extending 6d object pose estimators for stereo vision. *arXiv e-prints* (2024), arXiv–2402. 3

[PL19] PHUONG M., LAMPERT C.: Towards understanding knowledge distillation. In *International conference on machine learning* (2019), PMLR, pp. 5142–5151. 4

[PLK*24] PÖLLABAUER T., LI J., KNAUTHE V., BERKEI S., KUIJPER A.: End-to-end probabilistic geometry-guided regression for 6dof object pose estimation, 2024. URL: https://arxiv.org/abs/2409.11819, arXiv:2409.11819. 3

[SHL*23] SUNDERMEYER M., HODAŇ T., LABBE Y., WANG G., BRACHMANN E., DROST B., ROTHER C., MATAS J.: Bop challenge 2022 on detection, segmentation and pose estimation of specific rigid objects. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 2784–2793. 2

[SSF*22] SU Y., SALEH M., FETZER T., RAMBACH J., NAVAB N., BUSAM B., STRICKER D., TOMBARI F.: Zebrapose: Coarse to fine surface encoding for 6dof object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2022), pp. 6738–6748. 2, 3

[VA22] VADERA S., AMEEN S.: Methods for pruning deep neural networks. *IEEE Access 10* (2022), 63280–63300. 4

[WDH*23] WOO S., DEBNATH S., HU R., CHEN X., LIU Z., KWEON I. S., XIE S.: Convnext v2: Co-designing and scaling convnets with masked autoencoders. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2023), pp. 16133–16142. 4

[WMTJ21] WANG G., MANHARDT F., TOMBARI F., JI X.: Gdr-net: Geometry-guided direct regression network for monocular 6d object pose estimation. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition* (2021), pp. 16611–16621. 2, 3

[WY21] WANG L., YOON K.-J.: Knowledge distillation and student-teacher learning for visual intelligence: A review and new outlooks. *IEEE transactions on pattern analysis and machine intelligence 44*, 6 (2021), 3048–3068. 4

[XSNF18] XIANG Y., SCHMIDT T., NARAYANAN V., FOX D.: Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes. *Robotics: Science and Systems XIV* (2018). 2

[ZHW*23] ZHANG R., HUANG Z., WANG G., LIU X., ZHANG C., JI X.: Gpose2023: A modularized learning-based object pose estimator. *8th International Workshop on Recovering 6D Object Pose* (2023). 3