ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

# QUANTIZED NEURAL NETWORKS FOR 6D POSE ESTIMATION


by Ziqi Zhao

Saqib Javed
Assistant

Dr. Mathieu Salzmann
Thesis Supervisor


A thesis submitted in fulfillment of the requirements
for the degree of Master of Computer Science

***At***


Computer Vision Laboratory
School of Computer and Communication Sciences
EPFL


MASTER PROJECT REPORT

---

30th December 2022

# ACKNOWLEDGEMENTS

First of all, I would like to thank Mr. Saqib Javed for offering this master project and giving me suggestions and guidance throughout the whole project. All our meetings and discussions are quite beneficial and valuable to my research. I would also like to thank CVLab for providing the necessary computational resources for my experiments. I could hardly conduct my master project without them.

I especially want to acknowledge Dr. Mathieu Salzmann for his help during my research. He is the supervisor of my master project and also one of the supervisors of my first semester project. With one and half years of effort, the work of that semester project was extended and published on NeurIPS 2022. That project cultivated my research interests in network compression and adversarial robustness. It also motivated me to work on my master project which is also related to network compression.

Last but not least, I am grateful for the firm support from my family. They financed my whole master studies and helped me overcome the hardest time during my research. A sincere credit is also given to my friends, who also comforted me and brightened my daily life.

*Lausanne, 30th December 2022* Ziqi Zhao

# ABSTRACT

6D pose estimation is fundamental in many computer vision tasks such as augmented reality and satellite on-orbit services. Although the latest models can achieve promising accuracy to identify objects' poses, their heavy models are computationally inefficient and infeasible to be deployed on edge devices. Network quantization can solve this problem by compressing floating-point weights and activations to a finite set of values that can be expressed with lower bitwidths. At the same time, it will preserve the model's performance as much as possible. Network quantization has been widely studied on image classification tasks, yet little attention is paid to other applications, for example, 6D pose estimation. Since networks for 6D pose estimation are more modular than those for image classification, existing quantization algorithms might perform differently on them. Besides, pose estimation is a more challenging task than image classification because the model not only detects the object class but also estimates its 6D pose information. This project intends to study network quantization in the field of 6D pose estimation and explore how state-of-the-art quantization algorithms for image classification would perform in 6D pose estimation. Based on preliminary evaluations of the sensitivities of network modules, we find that the backbones in the networks are less resistant to quantization than other components. We then evaluate the mixed-precision quantization of a state-of-the-art method, HAWQ, and reveal that in a finer granularity, the layers at both ends of the network are usually more sensitive to quantization. We apply HAWQ to several 6D pose estimation networks using different mixed-precision quantization plans and discover that the performance degradation is more severe in 6D pose estimation. By using a multi-stage quantization strategy, we could achieve better performance on several datasets. Our experiments provide a deeper understanding of quantization and shed some light on a better quantization scheme for 6D pose estimation.

# Contents

# CHAPTER 1

# INTRODUCTION

Estimating object pose in images is fundamental to several vision-based tasks, including augmented reality [1], object grasping in robotic applications [2] and spatial services [3], [4]. Many augmented reality applications require high-precision pose information of objects in order to align the real world and virtual world; to grasp an object and manipulate it, a robotic arm needs to know the 6DoF information of that object before planning its movement; hundreds of kilometers above land, a reliable pose estimation algorithm can help spacecraft capture off-service satellites and clear the earth's orbit. Modern 6D pose estimation algorithms utilize neural networks to obtain better results than conventional matching algorithms. With the development of neural networks, the accuracy of 6D pose estimation keeps improving, whereas the network structures grow increasingly complicated and enormous. The powerful models require more computational power and more memory space, so it hinders the deployment on edge devices such as phones and spacecraft.

Network quantization is one of the popular methods to compress the model with little performance degradation. It restricts the model weights and/or activations within a small set of discrete values and minimizes the accuracy degradation simultaneously. By proper hardware implementation design, the compressed model can be further accelerated by TVM [5]. Quantization of neural networks has been well studied for image classification over the years and has shown its effectiveness [6]–[11], yet little attention is paid to 6D pose estimation. The networks for 6D pose estimation usually consist of a pre-trained backbone and a decoder (e.g., FPN [12], ASPP [13], etc.), which is more modular than those for image classification. The performance of network quantization under this architectural difference is hardly known and needs to be revealed. In addition, 6D pose estimation usually contains the classification of objects, making it a more challenging task than image classification.

In this project, we apply one state-of-the-art quantization algorithm, HAWQ [14]–[16], to multiple 6D pose estimation networks [3], [4], [17] and explore the sensitivity of different parts of the network and how they perform against different quantization levels. An initial evaluation demonstrates that the backbones are usually the most sensitive ones in 6D pose estimation models. We then propose a module-based mixed-precision quantization strategy to set different quantization levels to different modules. After analyzing HAWQ's mixed-precision quantization plan, we find that layers at both ends of the network are more sensitive to quantization than others. To stabilize the quantization process, we also propose to quantize the network in a multi-stage manner, namely quantizing part of the network in each stage, and fine-tuning the model after all layers have been quantized. Experiments show that multi-stage quantization might help the model find better solutions and preserve more performance after quantization. Although current experiment results are not strong enough to prove the feasibility of multi-stage quantization, we think that our findings suggest several directions for future work and that there are still gaps to fill in the field of quantization for 6D pose estimation.

The contributions of our work include the following aspects:

- We evaluate the Hessian-trace-based sensitivity of layers in state-of-the-art 6D pose estimation networks and find that in 6D pose estimation networks, some layers at the beginning or at the end are more sensitive to quantization, while other layers can tolerate more against quantization.
- We propose a multi-stage quantization strategy that quantizes part of the network at each quantization stage instead of quantizing the whole network at once.
- Our layerwise quantization strategy is capable of achieving better performance than a naive quantization that quantizes the network at once.

This report is organized as follows: we first review the literature on 6D pose estimation and network quantization in Chapter 2, then introduce the basics of quantization and all the quantization and 6D pose estimation algorithms we use in this project in Chapter 3. Our experiments setup and results will be demonstrated in Chapter 4. We conclude our report in Chapter 5.

# CHAPTER 2

# RELATED WORKS

In this chapter, we first review the literature on 6D pose estimation, including both dual-stage and single-stage methods. Then we introduce recent advances in network quantization. Specifically, we summarize works that focus on uniform and non-uniform quantization, quantization-aware training (QAT), post-training quantization (PTQ), and mixed-precision quantization (MPQ).

## 2.1 6D Pose Estimation

The latest 6D pose estimation methods can be classified into dual-stage ones and single-stage ones, based on whether the pose information is directly obtained from the networks. In particular, dual-stage 6D pose estimation algorithms first establish 3D-to-2D correspondence using a neural network, then use a PnP solver to estimate object pose; a single-stage pose estimation algorithm regresses the object pose in an end-to-end manner. We will review their recent advances below.

### 2.1.1 Dual-stage Pose Estimation

Recent dual-stage pose estimation methods aim to provide more reliable 3D-to-2D correspondences or additional uncertainty of the correspondences to improve the performance of the PnP solver. PVNet [18] regressed keypoint-pointing vectors for each pixel and voted for keypoint locations using these vectors. An uncertainty-aware PnP solver was utilized to account for uncertainties in the estimated 2D keypoint coordinates. SegDriven [19] proposed a segmentation-driven method to predict 3D-to-2D correspondences with confidence values. It showed that the combination of several local predictors can improve the robustness of correspondences. SurfEmb [20] adopted contrastive learning to establish continuous distributions of 3D-to-2D correspondences. Specifically, the model is trained to maximize the score of the correct pose among all pose hypotheses predicted by PnP-RANSAC. ZebraPose [17] designed an encoding method that represents hierarchical binary grouping information for object surface vertices. It also proposed a coarse to fine-grained training strategy to automatically update predicted encodings and focus on different granularities during different training phases.

### 2.1.2 Single-stage Pose Estimation

Compared with dual-stage pose estimation, a single-stage pose estimation algorithm can achieve higher efficiency and provide an end-to-end way to train the model and get pose estimations directly. Earlier works such as PoseCNN [21] designed a network that solves the semantic labeling, translation estimation,

and rotation regression simultaneously. Its novel loss function can also handle symmetry objects. However, earlier works could not outperform dual-stage pose estimation methods. SO-Pose [22] investigated this issue and found that one predicted correspondence might match several 6D pose estimations with similar errors, resulting in a sub-optimal solution. To overcome this problem, SO-Pose introduced self-occlusion information of objects and used the consistency between this information and predicted correspondence to select more accurate pose information. WDR [4] proposed a new network that can provide reliable pose estimations under extensive object depth ranges, which is crucial for satellite on-orbit services. During the inference phase, the predicted pose can be estimated by either a RANSAC-PnP solver or a learning-based solver [23]. CA-SpaceNet [3] extended WDR and adopted counterfactual analysis to remove the effect of the complicated background. In addition, it quantized the network and deployed it on FPGA. This reduced the inference latency, hence more hardware-friendly for deployment.

## 2.2 Network Quantization

Network quantization methods can be categorized by several criteria. From the perspective of quantizers, namely quantization functions, they can be divided into uniform quantization and non-uniform quantization; from the perspective of the quantization workload, there are quantization-aware training (QAT) and post-training quantization (PTQ) algorithms. In addition, some methods apply different bitwidths to different layers to make quantization adaptive to layer sensitivity. These methods are called mixed-precision quantization (MPQ). In this section, we will review recent advances in the above topics. A more comprehensive classification of network quantization can be found in [11].

### 2.2.1 Uniform and Non-Uniform Quantization

To quantize the weights and activations in a neural network, a simple function is a linear function that maps floating-point values to a finite set of values. This is called uniform quantization because both quantization steps and quantized values are uniformly distributed. It is simple to use and efficient to be deployed on the hardware, so it is commonly used in many works. BinaryConnect [24] was the initial work to quantize the weights to {-1, +1}, and it used the sign function to compress the weights. BinaryNet [25] extended this work and quantized both weights and activations. To further speed up the training process, DoReFa-Net [6] was proposed to support gradient quantization. The quantizers in these methods are simple linear quantizers, which can be formulated as Equation 3.13. Their scaling factors and zero points are fixed during training and not updated. A recent work LSQ-Net [26] made the scaling factors trainable and update them together with network parameters during quantization. LSQ+ [27] further made the zero points learnable to support quantization for negative activation values from Swish and other activation functions. Though linear quantizers are simple to learn, one drawback is that they might lose information when quantizing values from a bell-shaped distribution.

Some works [10], [28] proposed to learn the dynamics of the distribution and adjust the quantization range of each quantization level accordingly. Specifically, [10] learned the quantization ranges by extending straight-through estimator (STE) [29] to support the gradient of quantization ranges. This method can better capture the information of input values than uniform quantization and is also easier to be deployed on the hardware than non-uniform quantization. However, the trainable quantization ranges introduce a huge amount of computational cost and make it impossible to train on one GPU. [28] used a lookup table as a quantizer and formulated the quantization process as a lookup table learning process. It used temperatured softmax distribution to make the lookup table learnable, then introduced several strategies to speed up the convergence during training.

On the contrary, a non-uniform quantization function will map the input values to non-uniformly spaced

discrete values. For example, [30] used logarithmic distributions to represent values using power-of-two values; [31] further proposed an additive-of-two quantization to represent full-precision values using a sum of powers-of-two values. Due to the difficulty of hardware deployment, non-uniform quantization is less popular than uniform quantization and can only be accelerated on specially-designed hardware.

### 2.2.2 QAT and PTQ

Given a pre-trained model, we can either quantize the network and also update network parameters, or only do quantization without further fine-tuning. The former is Quantization-Aware Training (QAT), while the latter is Post-Training Quantization (PTQ). QAT methods [10], [14]–[16], [26]–[28], [32] can improve the performance of quantized networks, whereas they require to re-train the parameters of the network for many epochs and introduce much computational cost.

PTQ methods determine the quantized network parameters without re-training, so they are more efficient than QAT methods. As a trade-off, the compressed models from PTQ usually perform worse than those obtained from QAT. AdaRound [9] is one of the recent PTQ algorithms. It showed that a simple round-to-nearest quantizer can lead to sub-optimal results, and it formulated the rounding problem as a quadratic unconstrained binary optimization problem to reduce task loss.

### 2.2.3 Mixed-Precision Quantization

Based on the observation that different layers of the network have different sensitivity to quantization, setting different bitwidths for the layers is an effective approach to reduce performance degradation. Since modern neural networks are becoming deeper, the search space for layer bitwidths is vast and it is impossible to manually find the optimal policy. To overcome this issue, mixed-precision quantization (MPQ) was studied to allow an automatic process to search for the optimal bitwidths for network layers. HAQ [32] trained a reinforcement learning agent to find the optimal quantization policy under multiple constraints such as model size, inference latency, and energy. Though it can perform better than many methods under the same constraints, it is time-consuming for the RL agent to find the optimal policy.

HAWQ [14]–[16] adopted Hessian-based sensitivity to select the bitwidth. In HAWQ-v1 [14], the authors proposed to use Hessian eigenvalues of each layer to determine its bitwidth. The quantization process for the layers will be determined by the product of Hessian eigenvalues and the quantization error of each layer. One limitation of HAWQ-v1 is that although the layer with larger Hessian eigenvalues should have a larger bitwidth, the choice of that bitwidth is still made by domain experts. HAWQ-v2 [15] solved this problem by introducing the Pareto Frontier approach. It first replaced Hessian eigenvalues with average Hessian trace which is a better metric for sensitivity, then minimized the sum of products between Hessian trace and quantization error for each layer. Then given a target model size, HAWQ-v2 will automatically find the optimal quantization policy using the Pareto Frontier approach. HAWQ-v3 [16] extended this idea, introduced additional constraints (inference latency, bit operations), and formulated the search of bitwidths as an integer linear programming problem. After calculating the Hessian trace and quantization errors, the searching process can be finished within seconds, which is more efficient than HAQ and previous versions of HAWQ. Besides, it used dyadic numbers to enable integer-only inference where only integer addition, multiplication, and bit-shifting are performed. The quantized models from HAWQ-v3 are more hardware-friendly for deployment.

# CHAPTER 3

# PROBLEM STATEMENT AND METHODS

This chapter will review the state-of-the-art algorithms used in this project. Specifically, we will first summarize several 6D pose estimation methods [3], [4], [17] used in our experiments. Then we will introduce the basics of quantization, followed by the quantization algorithm HAWQ [14]–[16].

## 3.1 6D Pose Estimation

### 3.1.1 Preliminaries

For a given image $Im$ with a target object, a 6D pose estimation algorithm $f$ parameterized by $\theta$ can estimate a $3 \times 3$ rotation matrix $R$ and a $3 \times 1$ translation vector $T$

$$R, T = f(Im, \theta) \tag{3.1}$$

such that a 3D coordinate $P = (P_x, P_y, P_z)^T$ in the object frame can be mapped into the camera frame $u = (u_x, u_y)^T$. Given the camera's intrinsic matrix $K$, this mapping can be formulated as

$$\lambda \begin{bmatrix} u_x \\ u_y \\ 1 \end{bmatrix} = K(RP + T) \tag{3.2}$$

where $\lambda$ is a scaling factor.

### 3.1.2 WDR and CA-SpaceNet

WDR [4] used a DarkNet-53 [33] as the backbone, then appended a Feature Pyramid Network (FPN) [12] containing $K$ feature maps with increasing receptive fields. These feature maps will be fed to the network head to generate $C \times (2 \times 8 + 1)$ vectors indicating 8 2D offsets and 1 objectness indicator for $C$ object classes. The objective function of WDR includes the focal loss [34] $\mathcal{L}_{obj}$ and a pose regression loss $\mathcal{L}_{reg}$. Specifically, given the network output probability $p$ and the ground truth $y$, the focal loss is defined as:

$$\mathcal{L}_{obj}(p, y) = \begin{cases} -\alpha(1-p)^{\gamma} \log(p), & \text{if } y = 1 \\ -(1-\alpha)p^{\gamma} \log(1-p), & \text{otherwise} \end{cases} \tag{3.3}$$

$\alpha$ can alleviate the problem of class imbalance, while $\gamma \geq 0$, the focusing parameter, can help the model focus more on hard examples and less on easier ones.

Given the 3D re-projection error $e_i$ for the i-th image, the pose regression loss is defined as the sum of the smoothed L1 norm of the errors:

$$\mathcal{L}_{reg} = \sum_{i=1}^{N} sl_1(e_i) \tag{3.4}$$

WDR used the sum of $\mathcal{L}_{obj}$ and $\mathcal{L}_{reg}$ among all feature maps as the overall training loss:

$$\mathcal{L} = \sum_{k=1}^{K} (\mathcal{L}_{obj}^{(k)} + \mathcal{L}_{reg}^{(k)}) \tag{3.5}$$

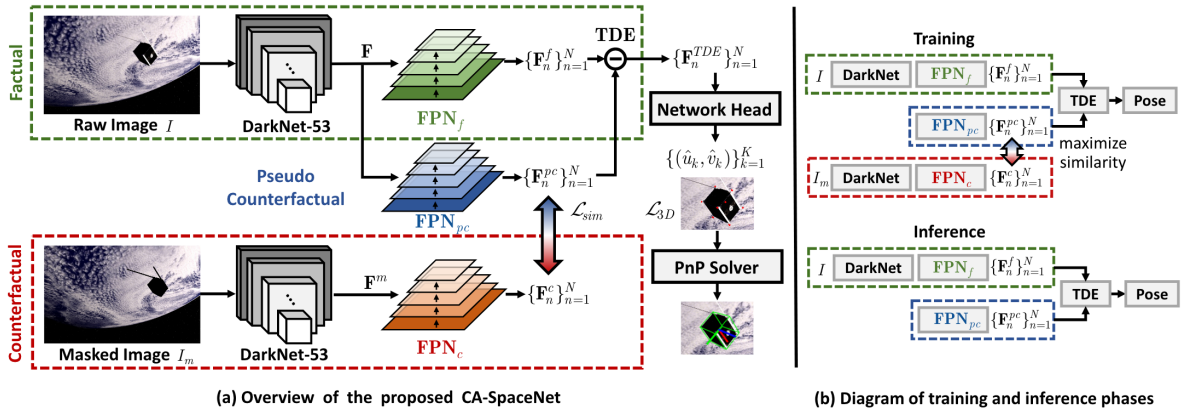


Figure 3.1: Overview of CA-SpaceNet [3]. The figure is extracted from the original paper.

CA-SpaceNet [3] extended WDR and proposed to use the principles from counterfactual analysis to train the model. An overview of CA-SpaceNet is shown in Figure 3.1. It contains three paths: the factual path which is the same as WDR, the counterfactual path containing the DarkNet-53 backbone and an FPN, and the pseudo-counterfactual path with only one FPN. The factual path takes raw images as input, whereas the counterfactual path takes images that only contain the background without the object. The goal of the counterfactual path is to learn the side effect of background and help the pseudo-counterfactual path learn this information. To achieve that, the authors introduced a similarity loss between the feature maps $\{\boldsymbol{F}_k^c\}$, $\{\boldsymbol{F}_k^{pc}\}$ of two FPNs in the counterfactual path and the pseudo-counterfactual path:

$$\mathcal{L}_{sim} = \sum_{k=1}^{K} sl_1(\boldsymbol{F}_k^{pc} - \boldsymbol{F}_k^c) \tag{3.6}$$

The overall loss function is the weighted sum of $\mathcal{L}_{obj}$, $\mathcal{L}_{reg}$ and $\mathcal{L}_{sim}$.

During inference, the counterfactual path will be removed, the pseudo-counterfactual path will be used to evaluate the side effect and remove it from the factual path.

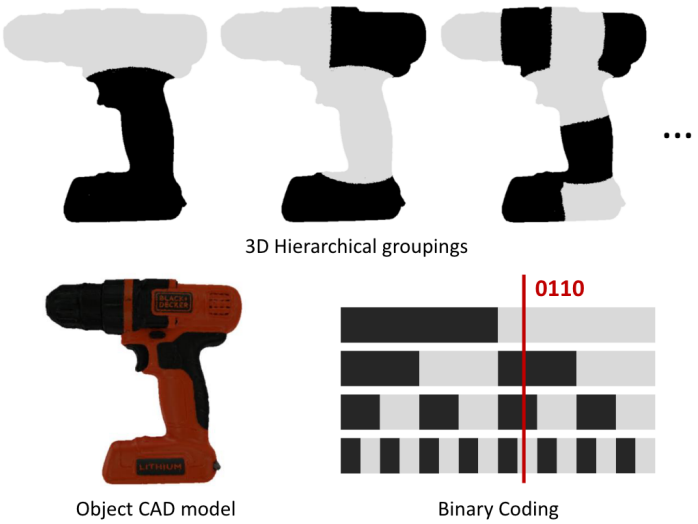


Figure 3.2: ZebraPose [17] hierarchically split object surface into binary groups. Then the one-to-one correspondence between object vertices and binary codes will be established and stored in a lookup table. The figure is extracted from the original paper.

### 3.1.3 ZebraPose

Instead of directly using 3D coordinates of object vertices as ground truth, ZebraPose [17] used a discrete descriptor that can efficiently encode the object vertices to binary codes in a hierarchical way. As Figure 3.2 demonstrated, ZebraPose hierarchically split the object vertices into binary groups and built a lookup table to store this code-to-vertex correspondence. It applied a modified version of DeepLabv3 [13] that added skipped connections and selected ResNet34 [35] as the backbone. The model will output a mask and a binary code vector for each pixel of the input image. Binary code vectors for pixels within the image masks will be used for training and inference.

There are two types of loss functions in ZebraPose: mask loss $\mathcal{L}_{mask}$ and hierarchical loss $\mathcal{L}_{hier}$. The former is defined as the L1 loss between the generated masks and ground truth ones. The latter can be computed by the weighted sum of Hamming distance between predicted codes and the ground truth. Specifically, given the predicted code probability $\hat{p} \in R^d$, the predicted code $\hat{b}$ can be calculated by rounding $\hat{p}$. To compute the Hamming distance between $\hat{p}$ and the ground truth $b$, one common practice is to apply a binary cross-entropy function before the calculation

$$Hamm(b, \hat{p}) = \sum_{i=1}^{d} b_i \log \hat{p}_i + (1 - b_i) \log(1 - \hat{p}_i) \tag{3.7}$$

Since each binary code vector encodes the object surface with different granularity, it is important to adjust their weights during training. ZebraPose defined a histogram for binary codes at each training step $t$

$$H_i(t) = avg(\lambda(b_i^t - \hat{b_i}^t) + (1 - \lambda)(b_i^{t-1} - \hat{b_i}^{t-1})) \tag{3.8}$$

Then a hierarchical loss is defined as

$$\mathcal{L}_{hier}(t) = \sum_{i=1}^{d} w_i(t) \cdot Hamm(b_i, \hat{p}_i) \tag{3.9}$$

where $w_i(t) = exp(\sigma \cdot min(H_i(t), 0.5 - H_i(t)))$. The overall loss function is the weighted sum of two loss functions

$$\mathcal{L} = \mathcal{L}_{mask} + \alpha \cdot \mathcal{L}_{hier} \tag{3.10}$$

# 3.2 Quantization

In this section, we will first review the basics of quantization, then introduce HAWQ in detail.

## 3.2.1 Preliminaries

For a supervised training process of a neural network parameterized by $\theta$ on the dataset $\mathcal{D} = \{(x_i, y_i)\}_{i=1}^{N}$, the objective should be:

$$\min_{\theta} \frac{1}{N} \sum_{i=1}^{N} \mathcal{L}(x_i, y_i, \theta) \tag{3.11}$$

Network quantization aims to compress the parameters $\theta$ under certain constraints such as model size and inference latency, and at the same time preserve as much accuracy as possible.

Generally, a quantizer is a function that maps floating-point numbers $r$ into a finite set of values:

$$\mathcal{Q}(r) = q_i, \text{ if } r \in [s_i, s_{i+1}) \tag{3.12}$$

where $\{s_i\}$ are quantization steps, and $\{q_i\}$ are quantization levels. For uniform quantization, a simple linear quantizer is defined as follows:

$$\mathcal{Q}(r) = \text{round}(\text{clamp}(\frac{r}{s}, q_{min}, q_{max})) - z \tag{3.13}$$

$$\overline{r} = r \cdot (s + z) \tag{3.14}$$

Here, $z$ is the zero-point, $s$ is the scaling factor, $q_{min}$ and $q_{max}$ define the clipping range. Given these parameters, the approximation of $r$ after quantization can be recovered by Equation 3.14. This recovery process is called dequantization.

The way to determine these parameters varies in different quantization algorithms. HAWQ uses symmetric quantization for weights, where the upper and lower clipping bounds are $q_{max} = \max(|r_{max}|, |r_{min}|)$, $q_{min} = -q_{max}$. Given the target bitwidth $k$, there are $2^k - 1$ possible values after quantization, ranging in $[1 - 2^{k-1}, 2^{k-1} - 1]$. The scaling factor is then determined by $s = \frac{q_{max}}{2^{k-1}-1}$. The zero point $z$ in symmetric quantization is set to zero.

Non-negative activations from ReLU can also be compressed by symmetric quantization when the target bitwidth is large. However, under extreme quantization (e.g. 2 bits), asymmetric quantization will provide a larger set of quantized values. In this case, $s$ will become $\frac{r_{max}-r_{min}}{2^k-1}$, $z = -\frac{r_{min}}{s} = -\frac{r_{min}(2^k-1)}{r_{max}-r_{min}}$. Due to the fact that activations are highly sensitive to quantization, we keep them at either 8 bits or 16 bits and only use symmetric quantization for them.

To remove floating-point operations and only use integer arithmetic in the inference phase, HAWQ pre-calculated the scaling factors of layers and used dyadic numbers to convert them into bit-shifting operations. For example, consider a layer with input $h$ and weight $w$, followed by a ReLU activation function. Both of them are quantized to $s_h q_h$ and $s_w q_w$, then the layer will output

$$a = s_h s_w (q_h * q_w) \tag{3.15}$$

where "*" denotes matrix multiplication or convolution. Denote the scaling factor of activation as $s_a$, then the quantized activation will become

$$q_a = \text{round}(\frac{a}{s_a}) = \text{round}(\frac{s_h s_w}{s_a}(q_h * q_w)) \tag{3.16}$$

$q_h * q_w$ can be carried out by integer multiplication and addition. However, calculating $\frac{s_h s_w}{s_a}$ directly requires floating-point operations. To convert this to integer arithmetic, HAWQ represents it in a dyadic number format:

$$DN(\frac{s_h s_w}{s_a}) = \frac{b}{2^c} \tag{3.17}$$

Then the scaling factor can be calculated by shifting $c$ bits of $b$.

For layers with residual connection (such as ResNet backbones), denote the output of the main branch as $m = s_m q_m$ and the output of the residual connection as $r = s_r q_r$, the quantized output after merging both branches will be

$$q_a = DN(\frac{s_m}{s_a})q_m + DN(\frac{s_r}{s_a})q_r \tag{3.18}$$

HAWQ also modified the BN folding [36] so that the parameters in the batch normalization layer and running statistics are all fixed during quantization. To be precise, the BN-folded convolution will be

$$\begin{aligned} Conv\_BN(h, w) &= \beta \frac{w * h - \mu}{\sigma} + \gamma \\ &= \frac{\beta}{\sigma} w * h + (\gamma - \frac{\mu}{\sigma}) \\ &= \overline{w} * h + \overline{b} \end{aligned} \tag{3.19}$$

Quantization will be performed on $\overline{w}$ and $\overline{b}$. For simplicity, $\overline{b}$ will use the scaling factor as $s_{\overline{w}} s_h$ so that the quantized bias can be added directly [1]

$$q_a = DN(\frac{s_{\overline{w}} s_h}{s_a})(q_{\overline{w}} * q_h + q_{\overline{b}}) \tag{3.20}$$

[1] Although HAWQ preserved $\overline{b}$ to be 32 bits regardless of the bitwidth of $\overline{w}$, we chose to quantize $\overline{b}$ to the same bitwidth as $\overline{w}$. This introduced additional performance degradation.

### 3.2.2 Module-based Mixed-Precision Quantization

Networks for 6D pose estimation are usually modular, and each module has its own objective. For example, the backbone in the model usually extracts features from the image, while a decoder will take features from multiple receptive fields and generate the pose-related information. Based on this observation, we could treat each module as a whole and set different quantization bits for different modules. We name this method as Module-based Mixed-Precision Quantization, or *Modular-based MPQ*. Table 3.1 shows an initial evaluation of the sensitivity of different modules in ZebraPose. When quantizing the decoder to 4 bits, the average performance degradation is 11%, while this value is 33% when quantizing the backbone to 4 bits. Therefore, a module-baed MPQ plan should assign larger bitwidths to the backbone and smaller bitwidths to the decoder.

| Object Name | Baseline | 4 bits Decoder | 4 bits Backbone |
|---|---|---|---|
| ape | 0.61 | 0.52 | 0.30 |
| can | 0.95 | 0.92 | 0.85 |
| cat | 0.60 | 0.48 | 0.37 |
| driller | 0.95 | 0.91 | 0.23 |
| duck | 0.58 | 0.21 | 0.27 |
| eggbox | 0.64 | 0.66 | 0.31 |
| glue | 0.85 | 0.81 | 0.75 |
| holepuncher | 0.73 | 0.49 | 0.18 |
| Average | 0.74 | 0.63 | 0.41 |

Table 3.1: Performance of ZebraPose models on a partial LMO dataset [37]. All the results are ADI-0.1d. The activations are all quantized to 16 bits.

### 3.2.3 Mixed-Precision Quantization in HAWQ

HAWQ provides another way of generating MPQ plans that consider Hessian-related sensitivity and other hardware constraints such as inference latency, model size, and bit operations [2]. HAWQ-v2 [15] proved that the average Hessian trace of weights is a good metric to evaluate their sensitivity to perturbations. Since computing the hessian matrix is a computation-heavy task, HAWQ used the Hutchinson algorithm [38] to approximate the Hessian trace without explicitly constructing the hessian matrix. Denote the Hessian matrix as $H$, and the identity matrix as $I$. Randomly sample a vector $z \in R^d$ whose elements are i.i.d. and drawn from Gaussian distribution $\mathcal{N}(0, 1)$. Then the trace of $H$ can be calculated by

$$Tr(H) = Tr(HI) = Tr(H\mathbb{E}[zz^T]) = \mathbb{E}[Tr(Hzz^T)] = \mathbb{E}[Tr(z^THz)] \tag{3.21}$$

After calculated the gradient $g_w = \frac{\partial \mathcal{L}}{\partial w}$, the multiplication $Hz$ can be obtained without knowing the hessian:

$$Hz = \frac{\partial g_w^T}{\partial w} z = \frac{\partial g_w^T}{\partial w} z + g_w^T \frac{\partial z}{\partial w} = \frac{\partial g_w^T z}{\partial w} \tag{3.22}$$

Hutchinson algorithm samples multiple $z$s and calculate the approximation of the Hessian trace as

[2]We only focus on the latest ideas from HAWQ-v3 [16]. The Hessian eigenvalue in HAWQ-v1 [14] and the Pareto Frontier approach in HAWQ-v2 [15] will not be discussed in this report.

$$Tr(H) \approx \frac{1}{n}\sum_{i=1}^{n} Tr(z_i^T H z_i) = \frac{1}{n}\sum_{i=1}^{n} Tr(z_i^T \frac{\partial g_w^T z_i}{\partial w}) \tag{3.23}$$

HAWQ further introduced the following metric to determine the overall sensitivity given the quantization plan $\{b_i\}_{i=1}^{n_w}$ for $n_w$ layers of weights

$$\Omega = \sum_{i=1}^{n_w} \Omega_i^{b_i} = \sum_{i=1}^{n_w} \overline{Tr}(H_i)\|\overline{r_i^{b_i}} - w_i\|_2^2 \tag{3.24}$$

where $\overline{Tr}(H_i)$ is the average hessian trace, $\overline{r_i^{b_i}}$ is the dequantized weight of $w_i$ from $b_i$ bits quantization.

The selection of $\{b_i\}_{i=1}^{n_w}$ is formulated as an integer linear programming (ILP) problem. Under $b_i$ bits quantization, denote the size of a matrix $w_i$ as $M_i^{b_i}$, the inference latency as $Q_i^{b_i}$ and the bit operations required for computation as $G_i^{b_i}$, the ILP can be formulated as follows

$$\text{Objective: } \min_{\{b_i\}_{i=1}^{n_w}} \sum_{i=1}^{n_w} \overline{Tr}(H_i)\|\overline{r_i^{b_i}} - w_i\|_2 \tag{3.25}$$

$$\text{Subject to: } \sum_{i=1}^{n_w} M_i^{b_i} \leq \text{Model Size Limit} \tag{3.26}$$

$$\sum_{i=1}^{n_w} G_i^{b_i} \leq \text{BOPs Limit} \tag{3.27}$$

$$\sum_{i=1}^{n_w} Q_i^{b_i} \leq \text{Latency Limit} \tag{3.28}$$

$G_i^{b_i}$ is calculated the same way as [39], i.e. using total multiply-accumulate operations $MAC_i$ and the bit precision of weights $w_i$ and activation $a_i$

$$G_i^{b_i} = b_{w_i} b_{a_i} MAC_i \tag{3.29}$$

Given the pre-computed $\Omega_i$ and the constraints, this ILP can be solved by PuLP[3] within seconds.

In this project, we removed the BOPs constraints and latency constraints, and only focus on the model size constraints for simplicity.

### 3.2.4 Multi-Stage Quantization

Quantizing the whole network usually suffers from instability. To stabilize the training process and avoid sub-optimal results, one feasible solution is to quantize the network in several stages instead of quantizing all the network layers at once. Multi-stage quantization was discussed in HAWQ-v1 [14], but this technique seems missing in later versions [15], [16]. Besides, little description was provided in the HAWQ-v1 paper. Based on the conclusion from HAWQ-v1 that layers with larger $\Omega$ should be quantized earlier than those with smaller $\Omega$, we proposed a *layerwise quantization* strategy.

[3]Available at: `https://github.com/coin-or/pulp`

We show our algorithm in Algorithm 1. In particular, for a model with weights $\{w_i\}_{i=1}^{n_w}$ in $n_w$ layers trained for $M$ epochs, we leave the last $F$ epochs for fine-tuning stage and divide the remaining training epochs into $n_w + 1$ equal parts, each with $I = \frac{(M-F)\times B}{n_w+1}$ iterations. Then we initialize all the weights to 32 bits and train for $I$ iterations. After every $I$ iterations, the layer with the largest $\Omega$ among the unquantized ones will be quantized. The learning rate before the fine-tuning stage will be fixed, whereas in the fine-tuning stage it will be a smaller one. In this project, we choose this learning rate to be 10% of the previous one.

Another form of multi-stage quantization is *modular quantization*. Instead of quantizing the network layer by layer, this method quantizes one module at one time. The modular quantization algorithm in this project shares the same logic as layerwise quantization, except that the order of quantization is not based on perturbations $\Omega$ but manually chosen.

Note that in this report, we refer "quantization plan" to the choices of bits $\{b_i\}_{i=1}^{n_w}$ given by the quantization schemes. Both module-based MPQ and HAWQ's MPQ will output quantization plans. "Quantization strategy" refers to the way of quantization given the quantization plans. Layerwise quantization and modular quantization are both quantization strategies. We denote the quantization strategy that quantizes the network at once as "naive quantization". The combination of a quantization plan and a quantization strategy is called "quantization policy".

**Algorithm 1** Layerwise Quantization

**Input:** training set $\mathcal{D}$ with batch size $B$, model $f$ with weights $\{w_i\}_{i=1}^{n_w}$ in $n_w$ layers, loss function $\mathcal{L}$, quantization plan $\{b_i^w\}_{i=1}^{n_w}$ for weights, perturbations for weights $\mathbf{\Omega} = \{\Omega_i\}_{i=1}^{n_w}$, activation bits $b_a$, total epochs $M$, fine-tuning epochs $F$, learning rate $lr$.
**Initialization:** Set quantization bits for weights to 32 bits, set quantization bits for activations to $b_a$ bits. Sort $\mathbf{\Omega}$ in descending order, denote the indices of sorted perturbations as $\boldsymbol{I}' = \{i_j\}_{j=1}^{n_w}$.
$I \leftarrow \frac{(M-F)\times B}{n_w+1}$ // Number of iterations for each quantization stage.
counter $\leftarrow 0$
**for** $m = 1, 2, \cdots, M$ **do**
  **if** $m = M - F$ **then**
    $lr \leftarrow 0.1 \times lr$ // Adjust the learning rate for the fine-tuning stage.
  **end if**
  **for** $b = 1, 2, \cdots B$ **do**
    counter $\leftarrow$ counter + 1
    **if** counter **mod** $I$ = 0 **and** counter $< (M - F) \times B$ **then**
      $j \leftarrow \frac{\text{counter}}{I}$
      set quantization bit for $w_{i_j}$ to $b_{i_j}^w$ // Quantize a new layer.
    **end if**
    Calculate the gradient $g$ of $\mathcal{L}$ on data batch $\mathcal{D}_b$ and update the weights $\{w_i\}_{i=1}^{n_w}$.
  **end for**
**end for**

# CHAPTER 4

# EXPERIMENTS

## 4.1 Experiment Setup

### 4.1.1 Dataset

We use Occluded-Linemod(LMO) [37] and SwissCube [4] datasets in our experiments. LMO was created from Linemod [40]. It contains 8 objects and includes images with heavy object occlusion. To facilitate the training, physically-based rendering (PBR) [41] images are used together with real images in LMO. SwissCube contains 50K rendered images for satellites in space. The distance between the satellite and the camera among images spans a wide range.

### 4.1.2 Training Setup

Unless specified below, we use the same setup (e.g. dataset splits, hyperparameter values) as ZebraPose, WDR, and CA-SpaceNet. In terms of the optimizer, ZebraPose uses Adam optimizer, while both WDR and CA-SpaceNet use SGD.

On SwissCube, WDR models are trained for 30 epochs with batch size 8. Under naive quantization, the initial learning rate is 0.01 and it drops to 10% after 20 and 25 epochs; under multi-stage quantization, the initial learning rate is set to 0.001 and drops to 10% in the fine-tuning stage starting from epoch 29. CA-SpaceNet uses the same setup as WDR, so we omit the details.

On LMO, WDR models are trained for 12 epochs with batch size 8. Under naive quantization, the initial learning rate is 0.005, and it drops to 10% after every 3 epochs; under multi-stage quantization, the initial learning rate is set to 0.01 and drops to 10% in the fine-tuning stage starting from epoch 11. ZebraPose models are trained for 20000 iterations with batch size 16. Under naive quantization, the initial learning rate is 0.0002, and it drops to 10% after every 5000 iterations; under multi-stage quantization, the initial learning rate is also set to 0.0002 and drops to 10% in the fine-tuning stage starting from approximately iteration 19000.

### 4.1.3 Evaluation Metrics

We evaluate the pose estimation accuracy by commonly-used ADI-0.1d [3], [4], [23]. If the average distance between the ground truth 3D points and the predicted 3D ones is less than 10% of the object diameter, it will count as successful. For symmetric objects, the matching between ground truth points

and predicted points are established to minimize their distance. ADI-0.1d is defined as the success rate among all the images. All the results in this report are ADI-0.1d accuracy.

## 4.2 Mixed-Precision Quantization

Our first quantization plan is a module-based MPQ plan. To be precise, we quantize WDR's backbone to 8 bits, FPN to 2 bits, and Head to 4 bits. For ZebraPose models, we quantize the backbone to 8 bits and the decoder to 4 bits. The weight compression ratio (WCR) for them is 4.1x and 4.6x respectively. Then we use these compression ratios to calculate HAWQ's model size limit in Equation 3.26 and ran ILP to find the HAWQ's MPQ plan. We will first interpret HAWQ's MPQ plan, then demonstrate the performance of quantized models.

### 4.2.1 Analysis of HAWQ MPQ Plans

Although HAWQ uses the total quantization error of the layer for ILP, it might not be a good variable for evaluation. We demonstrate the difference between the total quantization error $\Delta_b = \sum_i \|\overline{r_i^b} - w_i\|_2^2$ and the corresponding averaged values $\overline{\Delta}_b$ in Figure 4.1. Some middle layers have higher $\Delta$ because they have more parameters than other layers. However, we notice that some layers at the beginning and the end of the network show significantly higher average quantization errors. This is because weights in these layers have a larger range $r_{max} - r_{min}$, as displayed in Figure 4.2. HAWQ will use a much larger scaling factor $s$ to quantize these values, increasing quantization errors.

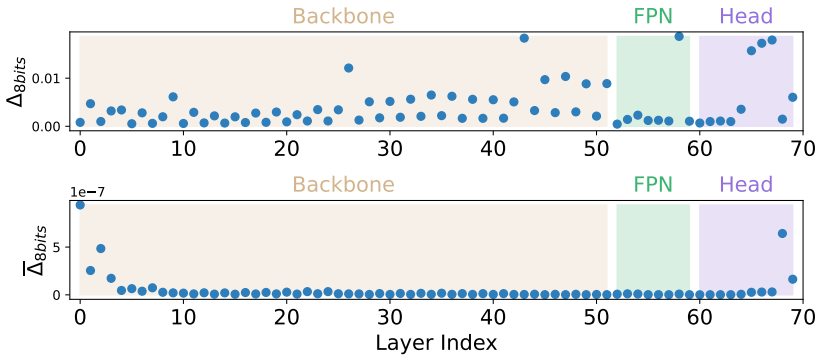


Figure 4.1: Comparison between the average and total quantization error of WDR's layers. The bitwidth is set to 8 bits. This pattern is occurring under all quantization bitwidths. ZebraPose also exhibits similar patterns. Shaded areas represent different modules of the network.

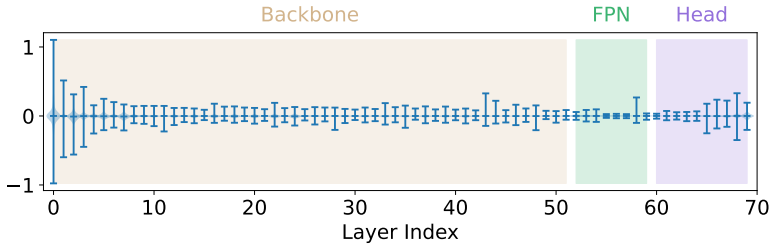


Figure 4.2: Weight distributions of layers in pre-trained WDR model.

Figure 4.3 provides a complete view of how average quantization errors change under different quantization

bitwidths. It can be seen that both WDR and ZebraPose models encounter larger quantization errors in the first and last few layers. Another observation is that the magnitude of quantization errors under different bitwidths varies a lot. We present later in Figure 4.4, 4.5 and 4.6 that this magnitude difference will largely affect the final perturbation $\Omega$ in HAWQ's MPQ plan.

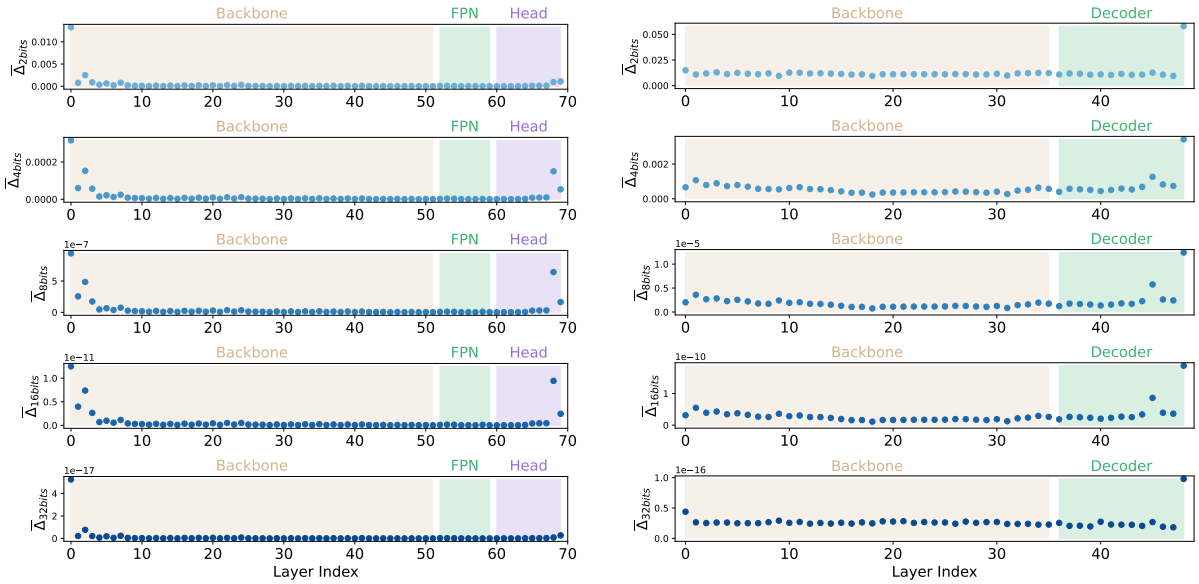


Figure 4.3: Average quantization error of layers of WDR (left) and ZebraPose (right) under different quantization bitwidths. Shaded areas represent different modules of the network.

We plot the average Hessian trace, HAWQ's MPQ plan, and resulting perturbations of the pre-trained WDR model in Figure 4.4. The first thing to notice is that most layers near the end of the network have higher average Hessian traces. These large values together with larger quantization errors induce a loose quantization level of 16 or 32 bits. Although the average Hessian trace of the first few layers is of small quantity, HAWQ also selects larger bitwidths for them because they produce large discrepancies after quantization. For other layers, HAWQ assigns 8 bits to most of them and 4 bits to those with relatively smaller quantization errors and average Hessian trace. Due to the magnitude difference discussed above, the resulting perturbation will be higher in layers with smaller bitwidths, and they will be quantized earlier in multi-stage quantization.

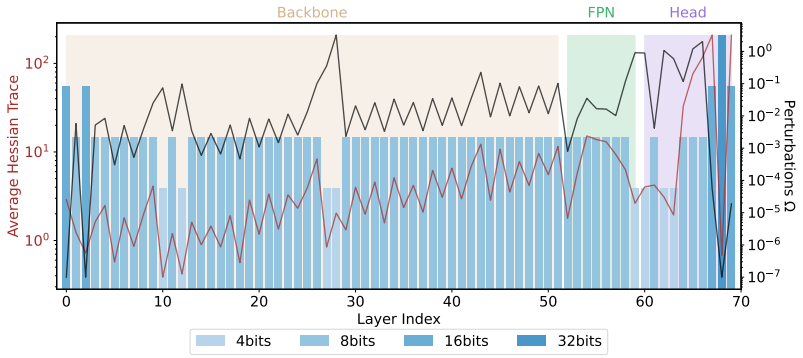


Figure 4.4: Average Hessian trace $\{\overline{Tr}(H_i)\}$, quantization plan $\{b_i\}$ and resulting perturbation $\{\Omega_i\}$ for layers in the WDR network. The target weight compression ratio is 4.1x. Shaded areas represent different modules of the network.

CA-SpaceNet has similar patterns to WDR. One difference is that many layers in $FPN$ and $FPN^{pc}$ have a small average Hessian trace. We show later in Table 4.2 that they are less sensitive than the head under extreme quantization.

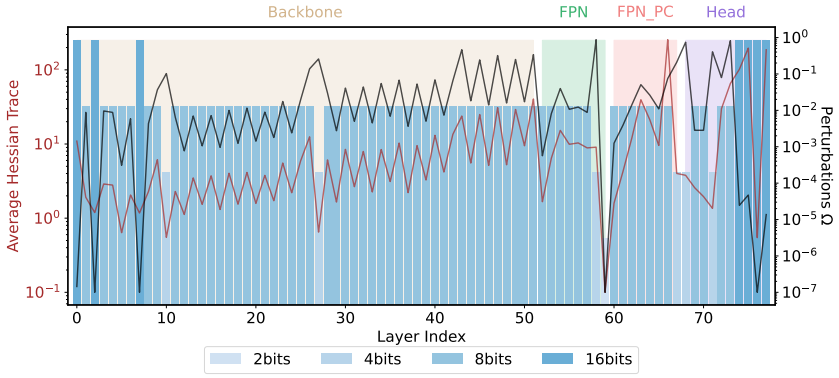


Figure 4.5: Average Hessian trace $\{\overline{Tr}(H_i)\}$, quantization plan $\{b_i\}$ and resulting perturbation $\{\Omega_i\}$ for layers in the CA-SpaceNet network. The target weight compression ratio is 4.1x. Shaded areas represent different modules of the network.

The pattern of Hessian trace becomes different in ZebraPose. As Figure 4.6 illustrates, large Hessian trace values can still be seen in the last few layers, but now this also happens in the first few layers. The first and last few layers still need to have larger quantization bitwidths. Layer 19-28 demonstrate small average Hessian trace, so they can be quantized to 4 bits; layer 29-35 has large quantization errors, so their bitwidths are assigned to 8.

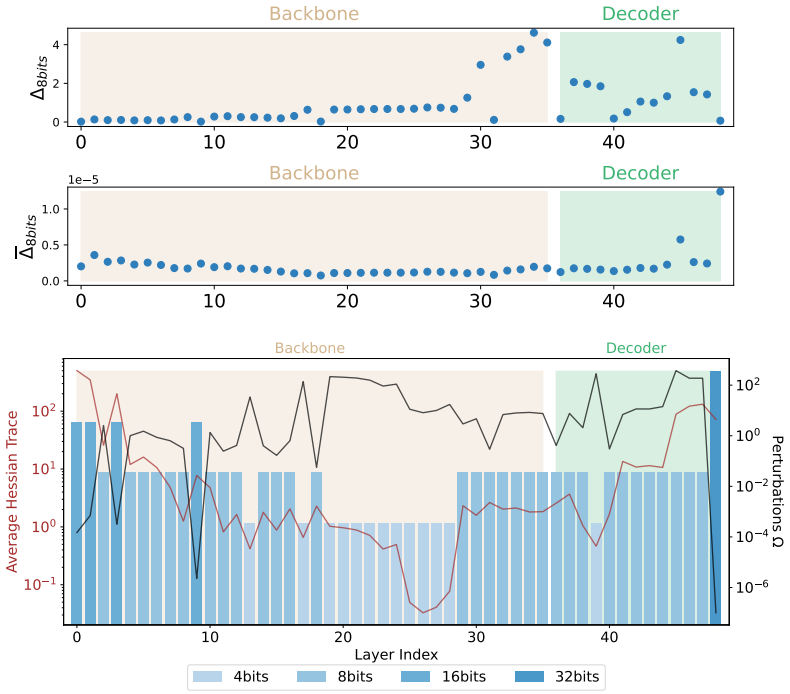


Figure 4.6: Top: Average quantization error and total quantization error of ZebraPose under 8 bits. Bottom: Average Hessian trace $\{\overline{Tr}(H_i)\}$, quantization plan $\{b_i\}$ and resulting perturbation $\{\Omega_i\}$ for layers in the ZebraPose network. The target weight compression ratio is 4.6x. Shaded areas represent different modules of the network.

To sum up, HAWQ tends to assign larger bitwidth to the first and last few layers, while there is no specific tendency for either middle layers or some modules. This is different from our module-based MPQ, which shows a strong preference to quantize the backbone to larger bits and other modules to smaller bits.

### 4.2.2 Performance on SwissCube

The performance of quantized WDR models on SwissCube is shown in Table 4.1. "b8f2h4" refers to the module-based MPQ plan that quantizes the backbone to 8 bits, FPN to 2 bits, and head to 4 bits. HAWQ's MPQ plan works better than the module-based one, with around 2% of improvements. It even performs better when the weight compression ratio is doubled. One reason to account for this is that HAWQ assigned large bits to the first and last few layers. To validate this idea, we manually fix the last layer for the module-based MPQ plan and quantize the model. This results in comparable performance to HAWQ's MPQ plan. For reference, a simple plan that quantizes all layers to 8 bits performs similarly to both MPQ plans, suggesting that MPQ plans might not matter much under the current compression ratio. Further experiments can be carried out under more extreme quantization levels.

| Quantization Policy | Weight Compression Ratio | ADI-0.1d(%) |
|---|---|---|
| Baseline | 1.0 | 78.95 |
| 8bits | 4.0 | 72.34 |
| b8f2h4 | 4.1 | 70.21 |
| b8f2h4 last fixed | 4.1 | 72.64 |
| mixed precision | 4.1 | 72.29 |
| mixed precision | 8.2 | 72.02 |

Table 4.1: Performance of WDR models on SwissCube dataset. The activations are all quantized to 8 bits. "b8f2h4" represents the module-based MPQ plan to quantize the backbone to 8 bits, FPN to 2 bits, and head to 4 bits. "mixed precision" means the MPQ plan given by HAWQ. "last fixed" means excluding the last layer from quantization. All the models are quantized using the naive quantization strategy.

| Quantization Policy | ADI-0.1d(%) |
|---|---|
| Baseline | 79.39 |
| $FPN^f$ 2bits | 78.78 |
| $FPN^{pc}$ 2bits | 79.29 |
| Both $FPN$s 2bits | 78.87 |
| Head 2bits | 53.38 |
| $FPN$s and Head 2 bits | 44.35 |
| Head 4bits | 77.14 |
| $FPN$s and Head 4 bits | 78.32 |

Table 4.2: Performance of CA-SpaceNet models on SwissCube dataset. The activations are all quantized to 8 bits. All the models are quantized using the naive quantization strategy.

We do not quantize the whole CA-SpaceNet in our experiments, because this model freezes the parameter update for its backbone. Therefore, we only quantize its FPNs and head and show the results in Table 4.2. Both $FPN^f$ and $FPN^{pc}$ are less sensitive to extreme quantization. Under 2-bit quantization, the ADI-0.1d degradation is less than 1%. On the contrary, when quantizing the head to 2 bits, the performance suffers from a significant reduction of around 26%. This is consistent when our analysis of Figure 4.5. After increasing the quantization bit of the head to 4 bits, the ADI-0.1d approaches the baseline with 2.25% of performance loss.

We also quantize both FPNs and the head together to 2 bits and 4 bits. Under 2-bit quantization, the performance decreases further to 44.35%. Surprisingly, under 4-bit quantization of FPNs and head, the ADI-0.1d is even larger than only quantizing the head to 4 bits. We plot their learning curves in Figure 4.7 and found that their learning curves are very similar, meaning that there is no significant difference between the performance of the two quantized models. The same trend happens in 2-bit quantization. One thing to notice is that the ADI-0.1d is fluctuating even at the end of training. This infers that the current training method might not be capable of the model to converge well.

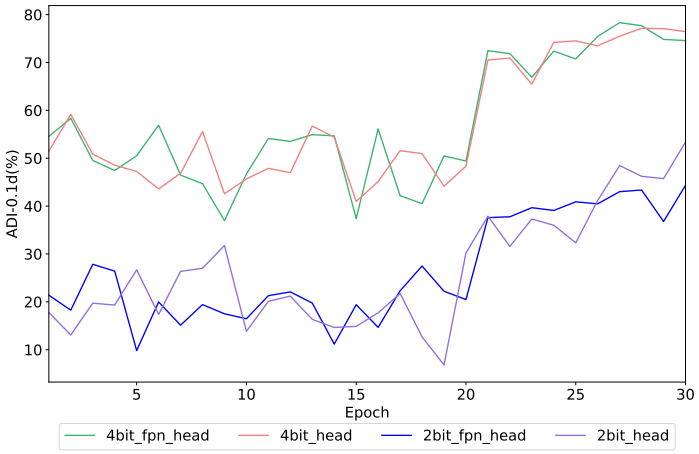


Figure 4.7: Learning curves of 2 and 4-bit quantization for a) both FPNs and Head; b) Head only.

### 4.2.3 Performance on LMO

| Policy | WCR | Overall | ape | can | cat | driller | duck | eggbox* | glue* | holepuncher |
|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 1.0 | 37.56 | 14.36 | 53.85 | 25.27 | 59.80 | 31.41 | 32.09 | 59.69 | 23.97 |
| 8bits | 4.0 | 29.83 | 18.8 | 39.19 | 8.34 | 38.55 | 21.00 | 37.36 | 46.40 | 29.01 |
| b8f2h4 | 4.1 | 16.08 | 8.80 | 16.16 | 7.75 | 15.98 | 16.10 | 22.55 | 27.24 | 14.05 |
| b8f2h4-LF | 4.1 | 16.39 | 11.03 | 19.06 | 6.82 | 18.20 | 14.52 | 25.36 | 29.01 | 7.11 |
| MP | 4.1 | 27.02 | 13.33 | 39.11 | 11.20 | 41.52 | 20.82 | 26.64 | 49.94 | 13.63 |
| MP | 8.2 | 12.40 | 8.547 | 12.01 | 8.00 | 12.60 | 12.77 | 6.21 | 22.37 | 16.69 |

Table 4.3: Performance of WDR models on LMO dataset. All the results are measured in %. All activations are quantized to 8 bits. "WCR" is the weight compression ratio. "b8f2h4" represents the module-based MPQ plan to quantize the backbone to 8 bits, FPN to 2 bits, and head to 4 bits. "MP" means the MPQ plan given by HAWQ, and "LF" means excluding the last layer from quantization. All the models are quantized using the naive quantization strategy. Objects marked "*" are symmetric objects.

Table 4.3 shows the performance of quantized WDR models on the LMO dataset. Since LMO has 8 object types and SwissCube only has one type of object, it is reasonable to find that the pre-trained network only achieves 37.56% overall ADI-0.1d. In the current settings, HAWQ's MPQ plan works significantly better than the module-based one. The module-based MPQ plan only gains 0.31% improvement after taking the last layer out of quantization, which is still much smaller than the HAWQ's MPQ plan. However, neither of these plans works better than a simple 8-bit quantization. One possible reason is that the learning rate decreases too much at the end of the training, hindering models from learning. As shown in Figure 4.8, the accuracy fluctuates a lot at the beginning of training and nearly stops increasing in the last half of training, so the performance increase in the first few epochs plays a crucial part in the final ADI-0.1d value. Further experiments can keep the initial learning rate for longer epochs before reducing it.

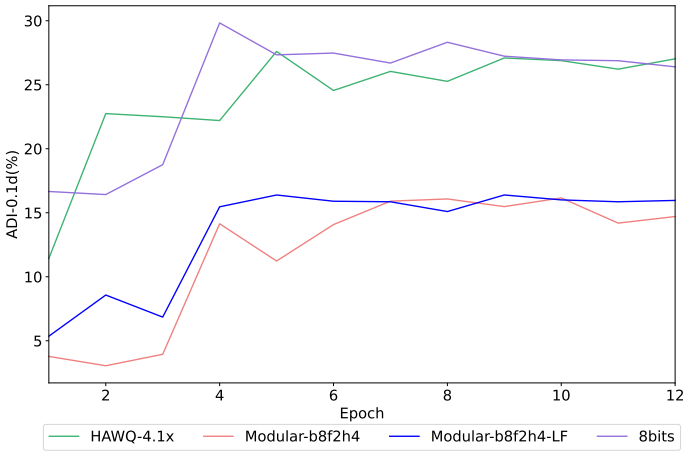


Figure 4.8: Learning curves of quantized WDR models on LMO dataset. "Modular-b8f2h4" means the module-based MPQ plan that quantizes the backbone to 8 bits, FPN to 2 bits, and head to 4 bits. "LF" means taking the last layer out of quantization.

ZebraPose trains one model for each object, so it performs better than WDR on the LMO dataset. We demonstrate the performance of quantized ZebraPose models in Table 4.4. Here we only compare two MPQ plans. Module-based MPQ plan works better than HAWQ's plan, with a minor improvement of 0.23% among all objects. By fixing the last layer and quantizing the rest, it can achieve an additional 0.69% increase. The last row of the table shows the performance of layerwise quantization, which will be discussed in the next section.

| Policy | WCR | Overall | ape | can | cat | driller | duck | eggbox* | glue* | holepuncher |
|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 1.0 | 76.93 | 57.9 | 95.0 | 60.6 | 94.8 | 64.5 | 70.9 | 88.7 | 83.0 |
| b8d4 | 4.6 | 71.35 | 54.49 | 93.95 | 57.24 | 91.10 | 44.72 | 67.46 | 85.88 | 75.95 |
| b8d4-LF | 4.6 | 72.04 | 54.92 | 93.95 | 57.24 | 91.85 | 43.28 | 68.51 | 87.51 | 79.09 |
| MP | 4.6 | 71.12 | 53.20 | 93.95 | 56.89 | 91.43 | 44.00 | 67.19 | 85.88 | 76.45 |
| MP-L | 4.6 | 72.54 | 57.34 | 93.53 | 56.28 | 91.10 | 55.28 | 65.17 | 86.70 | 74.88 |

Table 4.4: Performance of ZebraPose models on LMO dataset. All the results are measured in %. The activations are all quantized to 16 bits. WCR is the weight compression ratio. "b8d4" represents the module-based MPQ plan to quantize the backbone to 8 bits, and the decoder to 4 bits. "MP" means HAWQ's MPQ plan, and "LF" means excluding the last layer from quantization. Strategies ending with "L" stands for layerwise quantization, while others are using naive quantization. Objects marked with "*" are symmetric objects.

## 4.3 Multi-Stage Quantization

We first show the multi-stage quantization of WDR models on the SwissCube dataset in Table 4.5. Our layerwise quantization works best on HAWQ's MPQ plan, achieving 79.45% ADI-0.1d, much higher than the naive quantization. If we apply layerwise quantization in the reverse order, namely quantizing layers based on perturbations $\Omega$ in the ascending order, the performance is still better, but it is less than the original layerwise quantization. Compared with naive quantization using the "b8f2h4" plan, modular quantization ("b8f2h4 modular" in the table) yields an ADI-0.1d increase of 5.78%. By changing the quantization bits of activations from 8 bits to 16 bits, there can be an additional 0.53% improvement. Although applying layerwise quantization on the module-based MPQ plan can also improve the performance, it does not

show more advantages than modular quantization.

| Quantization Policy | Weight Compression Ratio | ADI-0.1d(%) |
|---|---|---|
| Baseline | 1.0 | 78.95 |
| b8f2h4 | 4.1 | 70.21 |
| mixed precision | 4.1 | 72.29 |
| b8f2h4 modular | 4.1 | 75.99 |
| b8f2h4 layerwise | 4.1 | 75.83 |
| b8f2h4 modular (16 bits activation) | 4.1 | 76.52 |
| reverse layerwise | 4.1 | 78.57 |
| layerwise | 4.1 | 79.45 |

Table 4.5: Comparison between naive quantization and multi-stage quantization of WDR models on SwissCube dataset. Unless specified, the activations are all quantized to 8 bits. "b8f2h4" represents the module-based MPQ plan to quantize the backbone to 8 bits, FPN to 2 bits, and head to 4 bits. "mixed precision" means the MPQ plan given by HAWQ. "Baseline", "b8f2h4" and "mixed precision" plans are using naive quantization, while the others use either modular or layerwise quantization.

We plot their learning curves in Figure 4.9. Among all quantization strategies, layerwise quantization using HAWQ's MPQ plan is the most stable one. When using reversed layerwise quantization, the model's performance will fluctuate in the middle of training, suggesting that some layers introduce a large perturbation to the model that is difficult to recover in a few epochs. This is consistent with what we have analyzed before. Regarding the quantized models using modular quantization, the one with 8 bits activation suffers from performance drop and fluctuation during training and recovers at the end. After using 16 bits activation, the fluctuation is less severe and the ADI-0.1d is easier to recover.

One interesting observation is that layerwise quantization using the module-based MPQ plan faces the largest performance degradation. Although it performs better than naive quantization, it shows no superiority over modular quantization, which seems unreasonable. There might exist a better way to select the order of quantization, for example, based on a modified version of perturbation $\Omega_i' = \overline{Tr}(H_i)\|\overline{r_i^{b_i}} - w_i\|_2^\alpha$ with a hyperparameter $\alpha$ to balance the focus between average Hessian trace and quantization error. In addition, the iterations and the learning rate for each quantization stage can be adjusted so that sensitive layers can be quantized for longer iterations and a smaller learning rate to recover the performance. These will leave as future work.

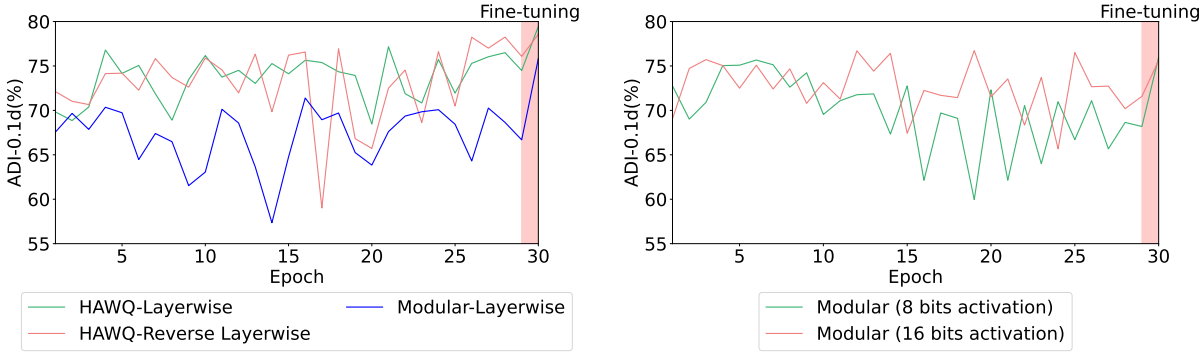


Figure 4.9: Learning curves of multi-stage quantization of WDR models on SwissCube dataset. Left: layerwise quantization. Right: modular quantization that quantizes a module at each stage.

We show in Table 4.6 the performance of layerwise quantization on WDR models with the LMO dataset. Similar to the previous table, layerwise quantization performs better than its reversed version. However,

none of them works better than quantizing the whole network at once. Their learning curves are illustrated in Figure 4.10. Different from Figure 4.9, the model's performance fluctuates hugely during the whole training process. We think that 12 epochs are insufficient for layerwise quantization to converge. Future experiments can set more training epochs to make layerwise quantization recover the model performance.

| Policy | WCR | Overall | ape | can | cat | driller | duck | eggbox* | glue* | holepuncher |
|---|---|---|---|---|---|---|---|---|---|---|
| Baseline | 1.0 | 37.56 | 14.36 | 53.85 | 25.27 | 59.80 | 31.41 | 32.09 | 59.69 | 23.97 |
| MP | 4.1 | 27.02 | 13.33 | 39.11 | 11.20 | 41.52 | 20.82 | 26.64 | 49.94 | 13.63 |
| MP-RL | 4.1 | 23.80 | 13.16 | 35.38 | 15.59 | 32.13 | 18.99 | 18.89 | 43.08 | 13.14 |
| MP-L | 4.1 | 26.08 | 10.51 | 38.94 | 12.3 | 32.95 | 9.62 | 38.64 | 32.34 | 33.31 |

Table 4.6: Comparison between naive quantization and multi-stage quantization of WDR models on LMO dataset. All the results are measured in %. The activations are all quantized to 8 bits. WCR is the weight compression ratio. "L" stands for layerwise quantization, "RL" stands for reverse layerwise quantization. "MP" means HAWQ's MPQ plan. "Baseline" and "MP" plans are using naive quantization. Objects marked with "*" are symmetric objects.

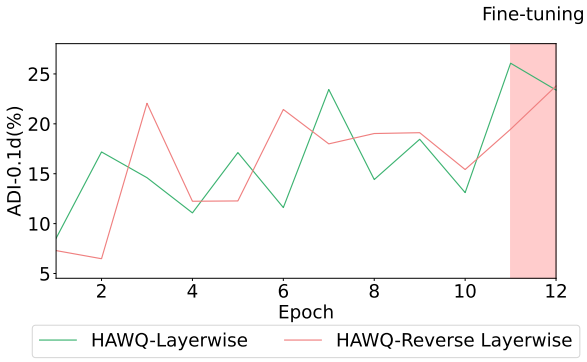


Figure 4.10: Learning curves of layerwise quantization of WDR models on LMO dataset.

Based on the above findings, we only conduct layerwise quantization on ZebraPose models and report their performance in Table 4.4. We find that layerwise quantization can improve the overall performance of 1.32%, achieving the highest one among all quantized ZebraPose models. However, the performance change of each object varies a lot. For example, object "duck" gets the largest improvement of 11.28%, while object "eggbox" receives a decrease of 2.02%. Their learning curves in Figure 4.11 reveal that fluctuations introduced by layerwise quantization seem to help models of hard objects to find better solutions and improve their performance. But since the learning rate in naive quantization decreases hugely at the end of quantization, it might also be possible that the current learning rate scheduler for naive quantization is not well defined for these models to recover the performance.

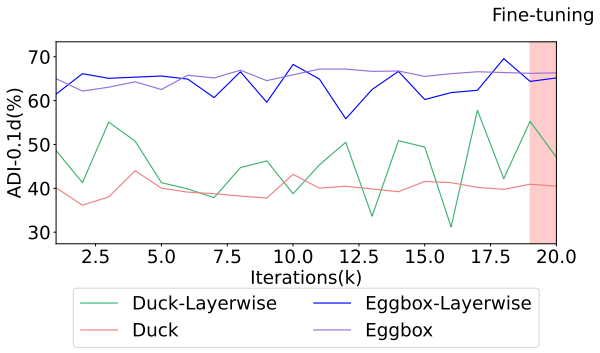


Figure 4.11: Learning curves of WDR models using different quantization strategies on LMO dataset.

# CHAPTER 5

# CONCLUSION

In this project, we explore network quantization for 6D pose estimation models. Initial studies suggest that the backbone of the network is more sensitive to other modules. We then develop a module-based MPQ accordingly. Further analysis of HAWQ's MPQ plan reveals that the beginning and ending layers are vulnerable to quantization and should not be quantized too much. To stabilize the quantization process, we propose a multi-stage quantization to quantize part of the network in each stage and fine-tune it after all layers are quantized. The order of quantization is based on HAWQ's perturbation $\Omega$. Experiment results show that multi-stage quantization can bring more performance to the quantized models in most cases. However, we are also aware that there is still room to improve current quantization algorithms.

As we discussed in Section 4, there are several directions for future work. Here are some additional interesting topics to be discovered:

- Apply HAWQ's MPQ plan to LSQ [26]. HAWQ only uses a simple linear quantizer and does not learn its parameters, whereas LSQ makes them learnable. We think the combination of the two will be more powerful to obtain a good-performing quantized model.
- Develop a smoother computation graph. Since quantization itself will introduce large parameter perturbation due to STE [29], the training process is not smooth and full of fluctuations. A smoother alternative for STE could be introduced to alleviate this issue, for example, G-STE in [10].

We hope this project could be beneficial and provide some insights to others who work on the same topic.

# REFERENCES


[1] E. Marchand, H. Uchiyama and F. Spindler, 'Pose estimation for augmented reality: A hands-on survey,' *IEEE transactions on visualization and computer graphics*, vol. 22, no. 12, pp. 2633–2651, 2015.

[2] G. Du, K. Wang, S. Lian and K. Zhao, 'Vision-based robotic grasping from object localization, object pose estimation to grasp estimation for parallel grippers: A review,' *Artificial Intelligence Review*, vol. 54, no. 3, pp. 1677–1734, 2021.

[3] S. Wang, S. Wang, B. Jiao *et al.*, 'Ca-spacenet: Counterfactual analysis for 6d pose estimation in space,' *2022 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2022.

[4] Y. Hu, S. Speierer, W. Jakob, P. Fua and M. Salzmann, 'Wide-Depth-Range 6D Object Pose Estimation in Space,' en, 2021, pp. 15 870–15 879. (visited on 10th May 2022).

[5] T. Chen, T. Moreau, Z. Jiang *et al.*, '{Tvm}: An automated {end-to-end} optimizing compiler for deep learning,' in *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*, 2018, pp. 578–594.

[6] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen and Y. Zou, 'Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,' *CoRR*, vol. abs/1606.06160, 2016.

[7] J. Choi, Z. Wang, S. Venkataramani, P. I.-J. Chuang, V. Srinivasan and K. Gopalakrishnan, 'Pact: Parameterized clipping activation for quantized neural networks,' *arXiv preprint arXiv:1805.06085*, 2018.

[8] D. Zhang, J. Yang, D. Ye and G. Hua, 'Lq-nets: Learned quantization for highly accurate and compact deep neural networks,' in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 365–382.

[9] M. Nagel, R. A. Amjad, M. Van Baalen, C. Louizos and T. Blankevoort, 'Up or down? adaptive rounding for post-training quantization,' in *International Conference on Machine Learning*, PMLR, 2020, pp. 7197–7206.

[10] Z. Liu, K.-T. Cheng, D. Huang, E. P. Xing and Z. Shen, 'Nonuniform-to-Uniform Quantization: Towards Accurate Quantization via Generalized Straight-Through Estimation,' en, 2022, pp. 4942–4952. (visited on 15th Nov. 2022).

[11] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney and K. Keutzer, 'A survey of quantization methods for efficient neural network inference,' in *Low-Power Computer Vision*, Chapman and Hall/CRC, pp. 291–326.

[12] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan and S. Belongie, 'Feature pyramid networks for object detection,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 2117–2125.

[13] L.-C. Chen, G. Papandreou, F. Schroff and H. Adam, 'Rethinking atrous convolution for semantic image segmentation,' *arXiv preprint arXiv:1706.05587*, 2017.

[14] Z. Dong, Z. Yao, A. Gholami, M. W. Mahoney and K. Keutzer, 'HAWQ: Hessian AWare Quantization of Neural Networks With Mixed-Precision,' 2019, pp. 293–302. (visited on 25th Jul. 2022).

[15] Z. Dong, Z. Yao, D. Arfeen, A. Gholami, M. W. Mahoney and K. Keutzer, 'HAWQ-V2: Hessian Aware trace-Weighted Quantization of Neural Networks,' in *Advances in Neural Information Processing Systems*, vol. 33, Curran Associates, Inc., 2020, pp. 18 518–18 529. (visited on 23rd Aug. 2022).

[16] Z. Yao, Z. Dong, Z. Zheng *et al.*, 'HAWQ-V3: Dyadic Neural Network Quantization,' en, in *Proceedings of the 38th International Conference on Machine Learning*, ISSN: 2640-3498, PMLR, Jul. 2021, pp. 11 875–11 886. (visited on 25th Jul. 2022).

[17] Y. Su, M. Saleh, T. Fetzer *et al.*, 'ZebraPose: Coarse To Fine Surface Encoding for 6DoF Object Pose Estimation,' en, 2022, pp. 6738–6748. (visited on 25th Jul. 2022).

[18] S. Peng, Y. Liu, Q. Huang, X. Zhou and H. Bao, 'Pvnet: Pixel-wise voting network for 6dof pose estimation,' in *CVPR*, 2019.

[19] Y. Hu, J. Hugonot, P. Fua and M. Salzmann, 'Segmentation-Driven 6D Object Pose Estimation,' 2019, pp. 3385–3394. (visited on 10th May 2022).

[20] R. L. Haugaard and A. G. Buch, 'SurfEmb: Dense and Continuous Correspondence Distributions for Object Pose Estimation With Learnt Surface Embeddings,' en, 2022, pp. 6749–6758. (visited on 25th Jul. 2022).

[21] Y. Xiang, T. Schmidt, V. Narayanan and D. Fox, 'Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes,' 2018.

[22] Y. Di, F. Manhardt, G. Wang, X. Ji, N. Navab and F. Tombari, 'SO-Pose: Exploiting Self-Occlusion for Direct 6D Pose Estimation,' en, 2021, pp. 12 396–12 405. (visited on 23rd Aug. 2022).

[23] Y. Hu, P. Fua, W. Wang and M. Salzmann, 'Single-Stage 6D Object Pose Estimation,' 2020, pp. 2930–2939. (visited on 10th May 2022).

[24] M. Courbariaux, Y. Bengio and J.-P. David, 'Binaryconnect: Training deep neural networks with binary weights during propagations,' *Advances in neural information processing systems*, vol. 28, 2015.

[25] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv and Y. Bengio, 'Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,' *arXiv preprint arXiv:1602.02830*, 2016.

[26] S. K. Esser, J. L. McKinstry, D. Bablani, R. Appuswamy and D. S. Modha, 'Learned step size quantization,' in *International Conference on Learning Representations*, 2020.

[27] Y. Bhalgat, J. Lee, M. Nagel, T. Blankevoort and N. Kwak, 'Lsq+: Improving low-bit quantization through learnable offsets and better initialization,' in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2020, pp. 696–697.

[28] L. Wang, X. Dong, Y. Wang, L. Liu, W. An and Y. Guo, 'Learnable Lookup Table for Neural Network Quantization,' en, 2022, pp. 12 423–12 433. (visited on 15th Nov. 2022).

[29] Y. Bengio, N. Léonard and A. Courville, 'Estimating or propagating gradients through stochastic neurons for conditional computation,' *arXiv preprint arXiv:1308.3432*, 2013.

[30] Z. Aojun, Y. Anbang, G. Yiwen, X. Lin and C. Yurong, 'Incremental network quantization: Towards lossless cnns with low-precision weights,' in *International Conference on Learning Representations,ICLR2017*, 2017.

[31] Y. Li, X. Dong and W. Wang, 'Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks,' in *International Conference on Learning Representations*, 2020.

[32] K. Wang, Z. Liu, Y. Lin, J. Lin and S. Han, 'HAQ: Hardware-Aware Automated Quantization With Mixed Precision,' 2019, pp. 8612–8620. (visited on 18th Dec. 2022).

[33] J. Redmon and A. Farhadi, 'Yolov3: An incremental improvement,' *arXiv*, 2018.

[34] T.-Y. Lin, P. Goyal, R. Girshick, K. He and P. Dollár, 'Focal loss for dense object detection,' in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2980–2988.

[35] K. He, X. Zhang, S. Ren and J. Sun, 'Deep residual learning for image recognition,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] B. Jacob, S. Kligys, B. Chen *et al.*, 'Quantization and training of neural networks for efficient integer-arithmetic-only inference,' in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 2704–2713.

[37] E. Brachmann, A. Krull, F. Michel, S. Gumhold, J. Shotton and C. Rother, 'Learning 6d object pose estimation using 3d object coordinates,' in *European conference on computer vision*, Springer, 2014, pp. 536–551.

[38] H. Avron and S. Toledo, 'Randomized algorithms for estimating the trace of an implicit symmetric positive semi-definite matrix,' *Journal of the ACM (JACM)*, vol. 58, no. 2, pp. 1–34, 2011.

[39] M. Van Baalen, C. Louizos, M. Nagel *et al.*, 'Bayesian bits: Unifying quantization and pruning,' *Advances in neural information processing systems*, vol. 33, pp. 5741–5752, 2020.

[40] S. Hinterstoisser, V. Lepetit, S. Ilic *et al.*, 'Model based training, detection and pose estimation of texture-less 3d objects in heavily cluttered scenes,' in *Asian conference on computer vision*, Springer, 2012, pp. 548–562.

[41] M. Denninger, M. Sundermeyer, D. Winkelbauer *et al.*, 'Blenderproc: Reducing the reality gap with photorealistic rendering,' in *International Conference on Robotics: Sciene and Systems, RSS 2020*, 2020.