UNIVERSIDADE FEDERAL DO PARANÁ

GABRIEL VITOR KLAUMANN GUBERT

AVALIAÇÃO DE DESEMPENHO DE UMA REDE NEURAL DE APRENDIZADO PROFUNDO QUANTIZADA EM POSIT PARA DETECÇÃO TRIDIMENSIONAL E CLASSIFICAÇÃO DE OBJETOS COM SINAIS DE RADAR E CÂMERA

CURITIBA

2024

GABRIEL VITOR KLAUMANN GUBERT

AVALIAÇÃO DE DESEMPENHO DE UMA REDE NEURAL DE APRENDIZADO PROFUNDO QUANTIZADA EM POSIT PARA DETECÇÃO TRIDIMENSIONAL E CLASSIFICAÇÃO DE OBJETOS COM SINAIS DE RADAR E CÂMERA

Requisito Parcial para o Título de Mestre em Engenharia pelo Programa de Pós-Graduação em Engenharia Elétrica (PPGEE) do Setor de Tecnologia da Universidade Federal do Paraná (UFPR).

Orientador: Prof. Luis Henrique Assumpção Lolis, DEng
Coorientador: Prof. Alessandro Zimmer, DEng

CURITIBA

2024

DADOS INTERNACIONAIS DE CATALOGAÇÃO NA PUBLICAÇÃO (CIP)
UNIVERSIDADE FEDERAL DO PARANÁ
SISTEMA DE BIBLIOTECAS – BIBLIOTECA DE CIÊNCIA E TECNOLOGIA

Gubert, Gabriel Vitor Klaumann
Avaliação de desempenho de uma rede neural de aprendizado profundo quantizada em POSIT para detecção tridimensional e classificação de objetos com sinais de radar e câmera / Gabriel Vitor Klaumann Gubert. – Curitiba, 2024.
1 recurso on-line : PDF.

Dissertação (Mestrado) - Universidade Federal do Paraná, Setor de Tecnologia, Programa de Pós-Graduação em Engenharia Elétrica.

Orientador: Luis Henrique Assumpção Lolis
Coorientador: Alessandro Zimmer

1. Redes neurais (Computação). 2. Veículos autônomos. I. Universidade Federal do Paraná. II. Programa de Pós-Graduação em Engenharia Elétrica. III. Lolis, Luis Henrique Assumpção. IV. Zimmer, Alessandro. V. Título.

Bibliotecário: Leticia Priscila Azevedo de Sousa CRB-9/2029

TERMO DE APROVAÇÃO

Os membros da Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação ENGENHARIA ELÉTRICA da Universidade Federal do Paraná foram convocados para realizar a arguição da Dissertação de Mestrado de **GABRIEL VITOR KLAUMANN GUBERT** intitulada: **Avaliação de desempenho de uma rede neural de aprendizado profundo quantizada em POSIT para detecção tridimensional e classificação de objetos com sinais de radar e câmera.**, sob orientação do Prof. Dr. LUIS HENRIQUE ASSUMPÇÃO LOLIS, que após terem inquirido o aluno e realizada a avaliação do trabalho, são de parecer pela sua APROVAÇÃO no rito de defesa.
A outorga do título de mestre está sujeita à homologação pelo colegiado, ao atendimento de todas as indicações e correções solicitadas pela banca e ao pleno atendimento das demandas regimentais do Programa de Pós-Graduação.

Curitiba, 28 de Agosto de 2024.

Assinatura Eletrônica
03/09/2024 09:45:50.0
LUIS HENRIQUE ASSUMPÇÃO LOLIS
Presidente da Banca Examinadora

Assinatura Eletrônica
03/09/2024 11:47:19.0
LEANDRO DOS SANTOS COELHO
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
03/09/2024 10:04:53.0
ANDREI CAMPONOGARA
Avaliador Interno (UNIVERSIDADE FEDERAL DO PARANÁ)

Assinatura Eletrônica
05/09/2024 09:38:36.0
SAMUEL BARALDI MAFRA
Avaliador Externo (INSTITUTO NACIONAL DE TELECOMUNICAÇÕES)

This thesis is dedicated to my family, my muse, and my friends.
To my family, for their support, unconditional love, and encouragement throughout my professional and academic journey. Your belief in me has been a constant source of strength and inspiration.
To my muse, for your endless patience, understanding, and love. Your presence has been a beacon of light during the most challenging times.
To my friends, for your camaraderie, support, and the joy you bring into my life. Your friendship has been invaluable in keeping me grounded and motivated.
Thank you all for being my pillars of support. This accomplishment is as much yours as it is mine.

ACKNOWLEDGEMENTS

This Master's Thesis is a result of a long-standing and fruitful international collaboration between the Graduate Program in Electrical Engineering (PPGEE) from the Department of Electrical Engineering (DELT) of the Federal University of Paraná (UFPR), in Brazil, and the International Automotive Engineering Master Course (IAE) from the Faculty of Electrical Engineering and Informatics at the Technische Hochschule Ingolstadt (THI), in Germany, and was only made possible with the support from the Bavarian Center for Applied Research and Technology with Latin America (AWARE) and the CARISSMA Institute of Safety in Future Mobility (C-ISAFE).

I would like to extend my deepest gratitude to Prof. Luis H. A. Lolis from the Department of Electrical Engineering (DELT) at the Federal University of Paraná (UFPR), and Prof. Alessandro Zimmer and Prof. Christian Birkner from the CARISSMA Institute of Safety in Future Mobility (C-ISAFE) at Technische Hochschule Ingolstadt (THI). Their unwavering support and guidance have been instrumental in the successful completion of this research, and, more broadly, throughout this journey across the oceans, from the Land of Araucarias to the Land of Poets and Thinkers.

Special thanks to Anderson de Lima Luiz and all my fellow colleagues, rese-archers, and professors from DELT and C-ISAFE who have contributed, in one way or another, to this Master's Thesis. Your assistance and support have been crucial in bringing this work to fruition.

“Any sufficiently advanced technology is indistinguishable from magic.“
— Arthur C. Clarke

RESUMO

A evolução da direção autônoma é influenciada pela convergência de tecnologias que moldam as capacidades dos sistemas de percepção. Este estudo reflete a interação entre aprendizado profundo, aceleradores de hardware especializados e representações numéricas avançadas, especificamente os Posits, no contexto do CenterFusion - um método de fusão intermediária para integrar dados de radar e câmera, desenvolvido para veículos autônomos. Posit é um tipo de representação e aritmética numérica em ponto flutuante cujos campos têm tamanhos variáveis, resultando em precisão não uniforme. Para otimizar o CenterFusion para sistemas embarcados, uma técnica de quantização de parâmetros e inferência de hardware usando números Posit foi desenvolvida, nomeada Posits4Torch e Posits4TorcHA. O principal objetivo deste estudo é avaliar a redução no uso de memória, o ganho em velocidade de processamento e a degradação de desempenho do CenterFusion, quando submetido à quantização com Posit, em termos de erros médios e do NuScenes Detection Score. Os resultados demonstram que os erros de inferência para o modelo emulado Deep PeNSieve em uma Unidade Central de Processamento (CPU) AMD EPYC 7413 de 24 núcleos e o modelo baseado em Posits4TorcHA em um Arranjo de Portas Lógicas Campo-Programáveis (FPGA) AMD Kria KV260 foram idênticos. O tempo de inferência do modelo baseado em FPGA de 3,49 segundos foi aproximadamente 1000 vezes menor que os 3194,69 segundos do modelo baseado em CPU, mas 300 vezes maior que o modelo baseado em Unidade de Processamento Gráfico (GPU) do PyTorch, executado em uma GPU NVIDIA GeForce RTX 3090. Além disso, as métricas de implementação em FPGA para o Arranjo de Unidades de Multiplicação e Acumulação (MAC) 2D 8x8 Posit no AMD Kria KV260 mostraram um consumo de energia de 2,939 W, 3,033 W e 3,090 W para precisões Posit de 6, 7 e 8 bits, respectivamente, destacando a eficiência de recursos da abordagem. Ao quantizar as cabeças de regressão do CenterFusion, uma precisão Posit de 8 bits para Posit-como-Armazenamento (PaS) resultou em uma degradação na Precisão Média (mAP) e no Escore de Detecção NuScenes (NDS) de, respectivamente, 0,7% e 0,5%. Para Posit-como-Aritmética (PaA) na FPGA, certas métricas como o Erro Médio de Velocidade (mAVE) e o Erro Médio de Translação (mATE) foram mais afetadas, com precisão de 8 bits levando a degradações na mAP e NDS de 9,8% e 7,7%, respectivamente, o que pode ser melhorado treinando o modelo com uma precisão Posit mais alta e subsequentemente quantizando para uma precisão Posit mais baixa usando Posits4Torch. Esta abordagem apresenta uma perspectiva promissora para otimizar o desempenho de redes neurais em sistemas de percepção e fusão de sensores para veículos autônomos. Ao melhorar a eficiência usando larguras de bits menores sem comprometer muito a precisão, esta pesquisa contribui para o desenvolvimento de soluções de IA de alta velocidade e mais eficientes em termos de energia para sistemas embarcados de veículos autônomos, demonstrando o progresso contínuo no campo da tecnologia de condução autônoma.

Palavras-chaves: posit; direção autônoma; aceleração de hardware.

ABSTRACT

The evolution of autonomous driving is influenced by the convergence of technologies shaping the capabilities of perception systems. This study reflects the interaction between deep learning, specialized hardware accelerators, and advanced numerical representations, specifically Posits, in the context of CenterFusion - a middle-fusion method for fusing radar and camera data designed for autonomous vehicles. Posit is a type of floating-point numerical representation and arithmetic whose fields have variable sizes, resulting in non-uniform precision. In order to optimize the CenterFusion for embedded systems, a parameter quantization and hardware inference technique using Posit numbers was developed, named Posits4Torch and Posits4TorcHA. The main objective of this study is to evaluate the reduction in memory usage, processing speed gain, and degradation of CenterFusion performance, when subjected to Posit quantization, in terms of the average errors and the NuScenes Detection Score. The results demonstrate that the inference errors for both the Deep PeNSieve-emulated model on an AMD EPYC 7413 24-Core Central Processing Unit (CPU) and the Posits4TorcHA-based model on an AMD Kria KV260 Vision Starter Kit Field-Programmable Gate Array (FPGA) were identical. The FPGA-based model's inference time of 3,49 seconds was nearly 1000 times lower than the 3194,69 seconds of the CPU model but 300 times higher than the PyTorch Graphics Processing Unit (GPU) model, which ran on a NVIDIA GeForce RTX 3090. Moreover, FPGA implementation metrics for the 2D 8x8 Posit MAC Unit Array on the AMD Kria KV260 showed a power consumption of 2,939 W, 3,033 W, and 3,090 W for Posit precisions of 6, 7, and 8 bits, respectively, highlighting the approach's resource efficiency. When quantizing CenterFusion's regression heads, an 8-bit Posit precision for Posit-as-Storage (PaS) resulted in a degradation in the Mean Average Precision (mAP) and NuScenes Detection Score (NDS) of, respectively, 0,7% and 0,5%. For Posit-as-Arithmetic (PaA) on the FPGA, certain metrics like Mean Average Velocity Error (mAVE) and Mean Average Translation Error (mATE) were more affected, with 8-bit precision leading to degradations in mAP and NDS by 9,8% and 7,7%, respectively, which can be improved by training the model with a higher Posit precision and subsequently quantizing to a lower Posit precision using Posits4Torch. This approach presents a promising perspective for optimizing neural network performance in perception and sensor fusion systems for autonomous vehicles. By improving efficiency using lower bit-widths without compromising too much accuracy, this research contributes to the development of more energy-efficient high-speed AI solutions for autonomous vehicle's embedded systems, demonstrating ongoing progress in the field of autonomous driving technology.

Key-words: posit. autonomous driving. hardware acceleration.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

AAE	Average Attribute Error
AD	Autonomous Driving
AF	Activation Function
AI	Artificial Intelligence
AMD	Advanced Micro Devices
AN	Artificial Neuron
AOE	Average Orientation Error
AP	Average Precision
API	Application Programming Interface
APU	Application Processing Unit
ASE	Average Scale Error
ASIC	Application-Specific Integrated Circuit
ATE	Average Translation Error
AV	Autonomous Vehicle
AVE	Average Velocity Error
BRAM	Block RAM
CMOS	Complementary Metal-Oxide-Semiconductor
CNN	Convolutional Neural Network
CPU	Central Processing Unit
CV	Computer Vision
DCN	Deformable Convolution Network
DL	Deep Learning
DLA	Deep Layer Aggregation

DMA	Direct Memory Access
DNN	Deep Neural Network
DSP	Digital Signal Processing
FCNN	Fully Connected Neural Network
FIFO	First-In First-Out
FM	Feature Map
FMA	Fused Multiply-Accumulate
FPGA	Field-Programmable Gate Array
GEMM	General Matrix Multiplication
GPU	Graphics Processing Unit
HA	Hardware Accelerator
HDL	Hardware Description Language
HLS	High-Level Synthesis
HTTP	Hypertext Transfer Protocol
IOU	Intersection over Union
IP	Intellectual Property
IPv4	Internet Protocol Version 4
IoT	Internet of Things
LIDAR	Light Detection and Ranging
LPE	Logarithm Posit Processing Element
LUT	Look-Up Table
MAC	Multiply-Accumulate
ML	Machine Learning
NDS	NuScenes Detection Score
NLP	Natural Language Processing
NN	Neural Network

NaN	Not-a-Number
PE	Processing Element
PIP	Package Installer for Python
PL	Programmable Logic
PTQ	Post-Training Quantization
PYNQ	Python Productivity for ZYNQ
PaA	Posit-as-Arithmetic
PaS	Posit-as-Storage
QAT	Quantization-Aware Training
RADAR	Radio Detection and Ranging
RAM	Random Access Memory
RL	Reinforcement Learning
RNN	Recurrent Neural Network
RTL	Register-Transfer Level
RTU	Reconfigurable Tensor Unit
ReLU	Rectified Linear Unit
RoI	Region of Interest
SIMD	Single Instruction, Multiple Data
SLR	Systematic Literature Review
SOM	System-On-Module
SOTA	State-of-the-Art
TCP	Transport Control Protocol
TL	Transfer Learning
TPU	Tensor Processing Unit
UDP	User Datagram Protocol
URAM	Ultra RAM

VHDL	Very High-Speed Integrated Circuit Hardware Description Language
VMA	Vector Multiply-Accumulate
mAAE	Mean Average Attribute Error
mAOE	Mean Average Orientation Error
mAP	Mean Average Precision
mASE	Mean Average Scale Error
mATE	Mean Average Translation Error
mAVE	Mean Average Velocity Error

LIST OF SYMBOLS

e	Euler’s Number
π	Pi

TABLE OF CONTENTS

1 INTRODUCTION

In the evolution of Autonomous Driving (AD), the convergence of technologies defines the capabilities of perception systems. Deep Learning (DL) forms the basis of these systems.

DL is a fundamental component of AD, 3D object detection and classification, and sensor fusion, playing a crucial role in the development of Autonomous Vehicle (AV) technology. It enables vehicles to process data from various sensors, recognize objects, interpret complex scenes, and adapt to diverse driving conditions. DL models, especially CNNs , enhance the accuracy and robustness of the 3D object detection and classification systems, facilitating precise object identification. Sensor fusion benefits from DL's ability to align and combine data from multiple sensors.

Based on this, Nabati e Qi (2021) introduces an intermediate sensor fusion method aimed at improving the accuracy of 3D object detection and classification in AD systems, called CenterFusion. By combining data from radar and camera sensors, the system aims to improve the estimation of depth, rotation, and velocity of objects in the environment. The performance evaluation of CenterFusion on the NuScenes dataset (Caesar et al., 2020) - a large-scale dataset and evaluation for AD - highlights its superior capability compared to existing camera-only algorithms, particularly in velocity estimation and overall detection accuracy. Furthermore, the complexity and computational demands of the NNs used in AVs introduce computational challenges, necessitating the integration of specialized HAs to achieve real-time processing.

HAs, such as ASICs and FPGAs , bring a new dimension to the optimization of DL models. These dedicated hardware solutions excel in parallel processing, enabling the fast execution of NN computations. ASICs, with their application-specific design, offer high performance and energy efficiency, while FPGAs provide a balance between adaptability and rapid prototyping capabilities. Both contribute to the deployment of efficient and high-performance perception systems in AVs.

Nevertheless, advanced numerical representations, such as Posit numbers and arithmetic (Gustafson; Yonemoto, 2017), emerge as potential game-changers. Posits offer an alternative to traditional floating-point and fixed-point representations, providing a dynamic range that adapts to the magnitude of the numbers represented. The emphasis on minimizing rounding errors and supporting a wider dynamic range aligns well with the demands of AD, where diverse and dynamic sensor inputs require precise numerical calculations.

Additionally, techniques such as pruning and quantization optimize DL models for deployment in HAs, ensuring efficiency both in computational terms and in the use of memory resources. Posit arithmetic, with its potential to enhance numerical precision and reduce bit-width, complements these efforts, further contributing to the overall efficiency of the perception system.

With this in mind, this work investigates the use of inference with Posit-quantized DNNs to optimize the efficiency of 3D object detection and classification in AD systems, evaluating these techniques applied to the CenterFusion middle-fusion method, which uses radar and camera data for 3D object detection and classification, and assessing its performance to demonstrate potential improvements in AV perception technologies.

1.1 OBJECTIVES

The General Objective of this work is to evaluate the impact of the Posit Quantization on the accuracy of a DL-based Camera and RADAR Middle-Fusion Method, the *CenterFusion*, when using Posit-as-Storage (PaS) on the GPU and Posit-as-Arithmetic (PaA) on the FPGA.

The Specific Objectives are defined as:

- Develop Posits4Torch, an FPGA-enabled Application Programming Interface (API) and Posit quantization customization for PyTorch;
- Quantize the Primary Regression Heads and Secondary Regression Heads of the CenterFusion using Posits through Posits4Torch;
- Accelerate on GPU the inference of the Posit-quantized Primary Regression Heads and Secondary Regression Heads of the CenterFusion using Posits4Torch and PaS;
- Accelerate on FPGA the inference of the Posit-quantized Primary Regression Heads and Secondary Regression Heads of the CenterFusion using Posits4Torch and PaA;
- Evaluate the accuracy of the Posit-quantized CenterFusion for 3D Object Detection and Classification, using the metrics from the NuScenes Detection Task, such as Mean Average Precision (mAP), Mean Average Translation Error (mATE), Mean Average Scale Error (mASE), Mean Average Orientation Error (mAOE), Mean Average Velocity Error (mAVE), Mean Average Attribute Error (mAAE), and NuScenes Detection Score (NDS).

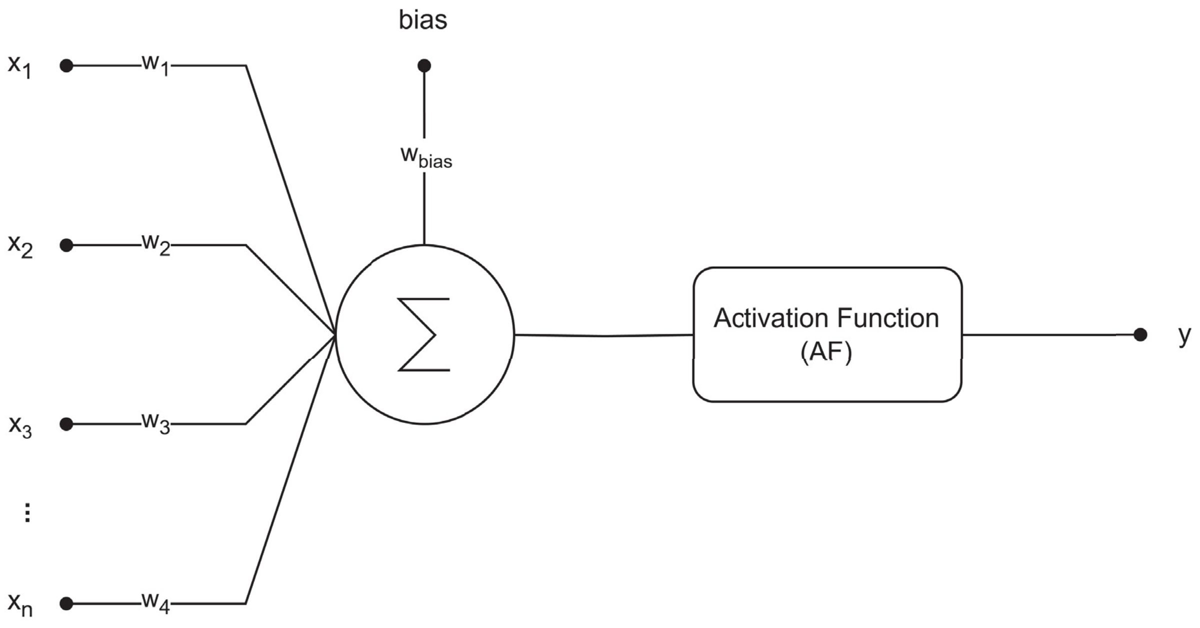
Finally, this document is structured as follows: the theoretical foundations and a review of relevant prior work are presented in CHAPTER 2 and CHAPTER 3, respectively. In CHAPTER 4, the methodology and experimental approach of this work are detailed, including the tools used. In CHAPTER 5, the findings of the study are displayed quantitatively and qualitatively, with visual aids and tables, and also provide insights into how these findings align or diverge from established knowledge. Finally, CHAPTER 6 encapsulates the research contributions and proposes directions for future investigations.

2 BACKGROUND

2.1 DEEP LEARNING NEURAL NETWORKS

A DNN is a computational model inspired by the human brain, composed of interconnected artificial neurons organized into layers. Its basic unit, the Artificial Neuron (AN), processes input data through weighted connections, introduces non-linearities through an Activation Function (AF), and generates an output (Vasilev; Slater, 2019). The basic structure of an AN can be visualized in the FIGURE 1.

FIGURE 1 – REPRESENTATION OF AN ARTIFICIAL NEURON.



SOURCE: The Author.

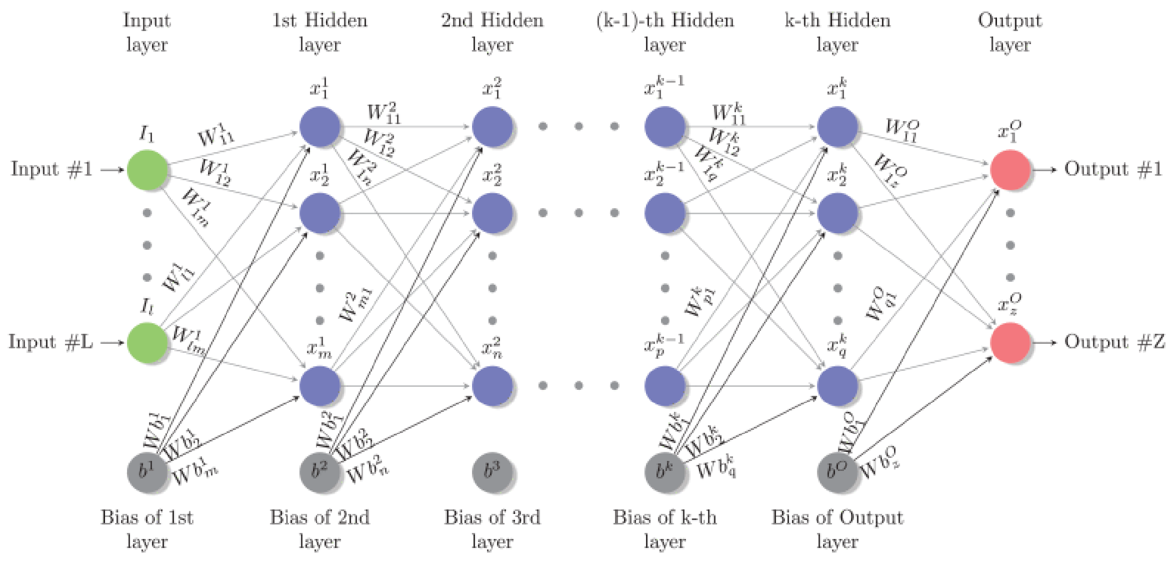
For an AN with n inputs (see FIGURE 1), the output is given by EQUATION 2.1,

$$y = AF\left(\sum_{i=1}^{n} w_i \cdot x_i + w_{bias} \cdot bias\right) \tag{2.1}$$

where y is the output, AF is the AF, w_i is the weight assigned to input x_i, and w_{bias} is the weight assigned to the bias, $bias$, of this neuron, usually equal to 1. The bias of an artificial neuron is associated with the activation potential analogous to a biological neuron.

The layers include the input layer for initial data, hidden layers for complex calculations, and the output layer for final results, as shown in the FIGURE 2.

FIGURE 2 – MODEL OF A FULLY CONNECTED NEURAL NETWORK (FCNN).



SOURCE: Belabed et al. (2021)

For a given layer, k, of the Fully Connected Neural Network (FCNN) in the FIGURE 2, the output, x_i^k, of the i-th neuron in that layer is given by the EQUATION 2.2,

$$x_i^k = AF^k \left(\sum_{j=1}^{n_{k-1}} w_{ji} \cdot x_j^{k-1} + b^k \cdot wb_i^k \right) \tag{2.2}$$

where n_{k-1} is the number of neurons in layer $k-1$, w_{ji} is the weight assigned to the connection from the output of neuron j in layer $k-1$, given by x_j^{k-1}, to the input of neuron i in layer k, b^k is the bias of layer k, wb_i^k is the weight assigned to the bias of layer k for neuron i, usually equal to 1, and AF^k is the AF for layer k.

AFs, such as Sigmoid (EQUATION 2.3) or Rectified Linear Unit (ReLU) (EQUATION 2.4), add crucial non-linearities to learn intricate patterns and also help NNs avoid saturation of values flowing between layers. However, for this, AFs require adequate precision of numerical formats used to represent the parameters of the NN.

NNs can take various forms, such as FCNNs for standard tasks, CNNs for Image Processing, and RNNs for sequential data (Vasilev; Slater, 2019).

$$\sigma(x) = \frac{1}{1+e^{-x}} \tag{2.3}$$

$$ReLU(x) = \max(0, x) \tag{2.4}$$

Weights and biases, parameters associated with connections, are adjusted during training to minimize the difference between predicted and actual results. Training involves forward propagation and backpropagation, with the latter adjusting the NN parameters based on the error between predicted and ground-truth results.

The flexibility of NNs allows them to excel in various applications, including image and voice recognition, language translation, and AVs. Their ability to automatically learn complex patterns makes them a powerful tool in the field of Machine Learning (ML).

In the context of AVs, NNs act as the brain behind the vehicle's perception and decision-making. They analyze visual data, identify objects, and merge sensor information for a comprehensive understanding of the environment. Through semantic segmentation and predictive models like RNNs, NNs facilitate trajectory planning, for example. Reinforcement Learning (RL) guides decision-making, adapting behavior based on rewards. NNs also contribute to mapping, localization, and driver monitoring, ensuring a holistic approach to safe and adaptive AVs. NNs are the driving force behind the vehicle's ability to navigate, interpret, and respond intelligently to the dynamic road environment.

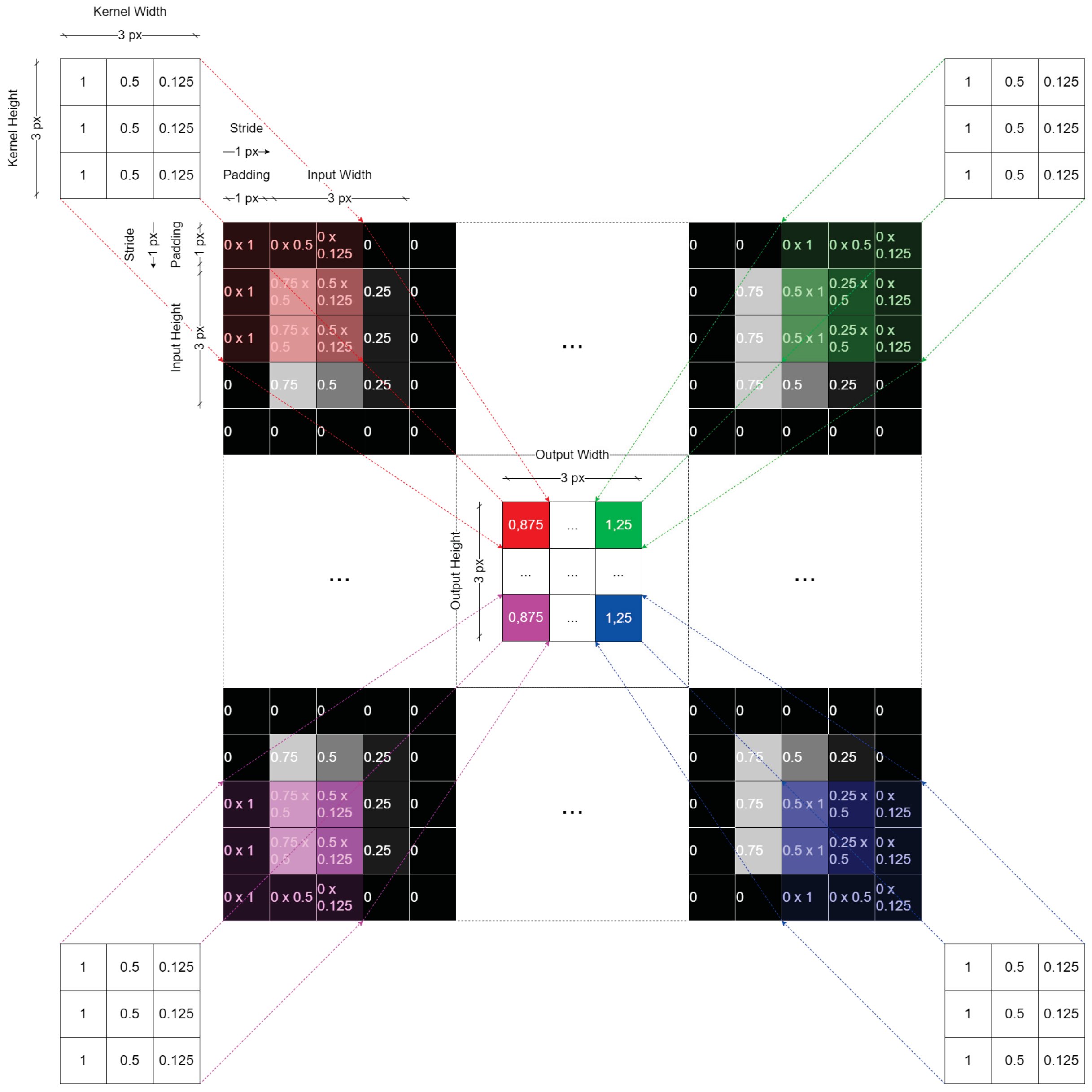
2.1.1 Deep Learning Convolutional Neural Networks

CNNs are a specialized class of artificial NNs designed for visual data processing, especially in tasks like image classification and object recognition. The architecture of a CNN is inspired by the human visual system and comprises several crucial components (Goodfellow et al., 2016).

A convolutional layer is a fundamental component within CNNs, mainly employed in tasks related to image analysis and recognition (Vasilev; Slater, 2019). It operates through a set of key elements: filters or kernels, convolution operation, stride, padding, and handling multiple channels. Filters are small trainable matrices designed to detect specific features, such as edges or textures in the input data. The convolution operation involves sliding these filters over the input, performing element-wise multiplications and summations to create output FMs , as shown in FIGURE 3. The stride controls the step size of this operation, affecting the spatial dimensions of the output FMs, while padding can be applied to preserve or adjust these dimensions (Vasilev; Slater, 2019; Goodfellow et al., 2016). It is important to note that convolutional layers can easily process multi-channel data, such as color images, by applying filters independently to each channel and combining the results, as visualized in FIGURE 4. This hierarchical feature extraction process enables CNNs to recognize increasingly complex and abstract patterns, making them invaluable for tasks like image classification and object detection and classification in 2D or 3D.

The equation for a convolutional layer in a CNN can be represented as follows:

FIGURE 3 – EXAMPLE OF 2D CONVOLUTION OPERATION ON A GRAYSCALE IMAGE.



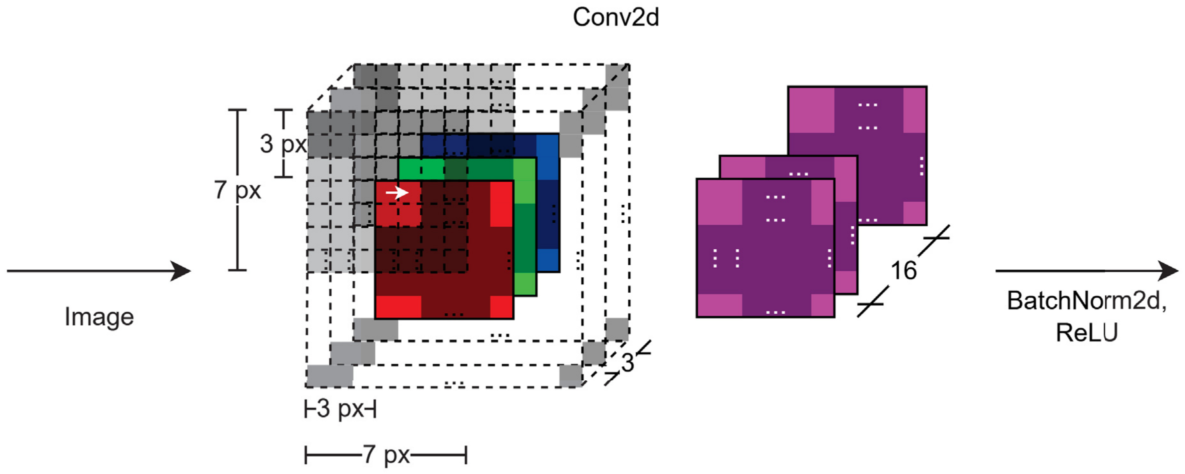
SOURCE: The Author.
NOTE: In the figure, the 2D Convolution Operation is characterized by having 1 Input Channel (Grayscale Image) of 3x3 pixels, 1 Output Channel, 1-pixel padding, 1-pixel stride, and 3x3 pixel kernel.

$$Y(i,j,k) = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} \sum_{l=0}^{L-1} (X(i+m, j+n, l) * W(m,n,l,k)) + b(k) \tag{2.5}$$

Where:

- $Y(i,j,k)$ is the value of the output FM at position (i, j) in channel k.
- $X(i+m, j+n, l)$ is the value of the input FM at position (i+m, j+n) in channel l.

FIGURE 4 – EXAMPLE OF APPLYING A CONVOLUTIONAL LAYER TO A 3-CHANNEL COLOR IMAGE (RED, GREEN, AND BLUE).



SOURCE: The Author.
NOTE: In the figure, the Convolutional Layer is characterized by having 3 Input Channels, 16 Output Channels, 3-pixel padding, 1-pixel stride, and 7x7 pixel kernels.

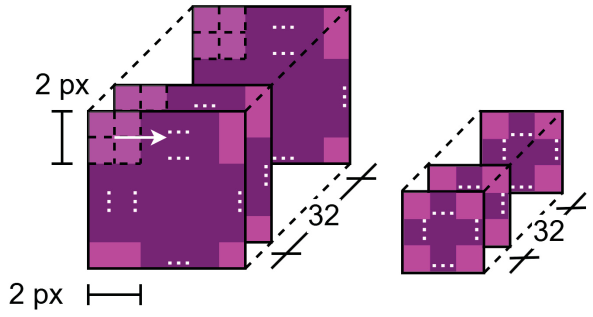
- $W(m, n, l, k)$ is the value of the convolutional filter at position (m, n) in input channel l and output channel k.
- $b(k)$ is the bias term associated with output channel k.

The summation is performed over all possible positions (m, n) within the filter size and all input channels l.

In practice, this equation can be implemented using efficient convolution operations, such as the "im2col"technique, which converts the input and filter into matrices and then performs matrix multiplication followed by adding the bias term to obtain the output FM (Lee et al., 2023). This allows for faster and optimized computations in DL frameworks.

Pooling Layers in CNNs are used to reduce the input data size while preserving important information. There are two main types: Max Pooling, which selects the maximum value from small regions, and Average Pooling, which computes the average (Vasilev; Slater, 2019). Pooling Layers help create hierarchical representations of features in CNNs, with initial layers capturing fine details and deeper layers recognizing larger patterns. They introduce translation invariance, allowing the NN to identify features regardless of their exact location. Pooling layers have hyperparameters like window size, stride, and padding that influence their behavior. An example of a Pooling layer, specifically Max Pooling, can be seen in FIGURE 5. Essentially, pooling layers make CNNs computationally efficient and effective for tasks like image recognition.

FIGURE 5 – EXAMPLE OF APPLYING A MAX POOLING LAYER TO INPUT FEATURE MAPS.



SOURCE: The Author.
NOTE: In the figure, the Max Pooling Layer is characterized by having 32 Input Channels, 32 Output Channels, 0-pixel padding, 2-pixel stride, and 2x2 pixel Window Size.

The equation for a Max Pooling layer in a CNN is:

$$Y(i,j,k) = \max_{m,n}(X(i \cdot s + m, j \cdot s + n, k)) \tag{2.6}$$

For an Average Pooling layer, it is:

$$Y(i,j,k) = \frac{1}{w}\sum_{m,n} X(i \cdot s + m, j \cdot s + n, k) \tag{2.7}$$

In these equations, X represents the input FM, Y represents the output FM after pooling, (i, j, k) are the indices in the output FM, and (m, n) are the indices within the pooling window. s is the stride with which the pooling window moves, and w is the pooling window size (e.g., 2x2 for typical max or average pooling).

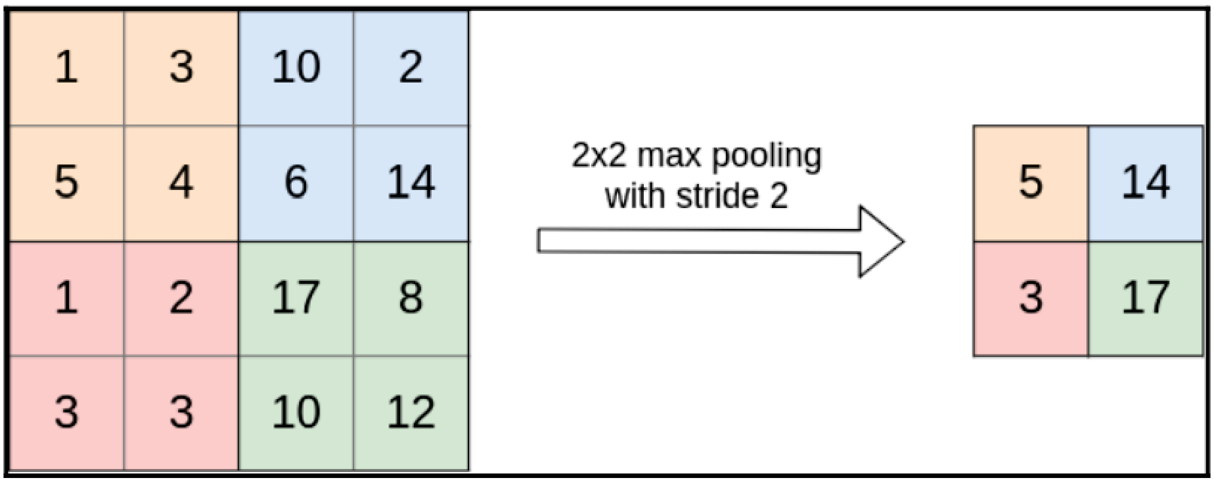
These equations describe how Max Pooling and Average Pooling operations are applied to the input FM to produce the output FM in a pooling layer of a CNN. A visual representation of both operations can be seen in FIGURE 6 and FIGURE 7.

AFs, such as the popular ReLU, introduce non-linearities into the CNN. This is essential for the model to understand intricate relationships in the data, contributing to its ability to comprehend complex visual patterns.

Fully connected layers (see FIGURE 2) connect all neurons in one layer to all neurons in the next, consolidating high-level features for final classification. The flattening step transforms the multidimensional output of previous layers into a one-dimensional vector suitable for these fully connected layers.

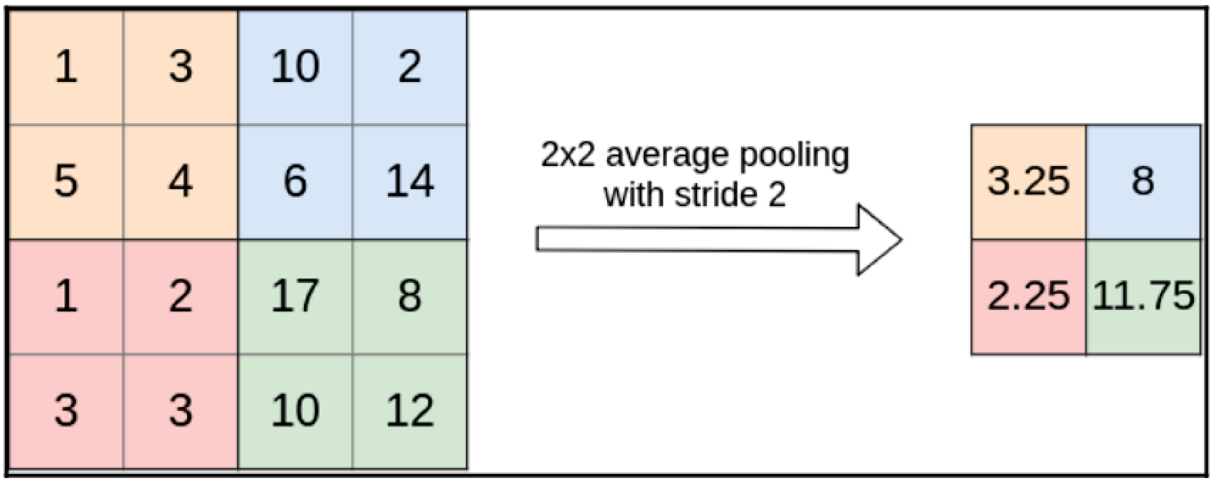
In the context of AD, CNNs are essential for interpreting sensor data. They excel at object detection and classification in 2D or 3D, recognizing pedestrians and vehicles,

FIGURE 6 – EXAMPLE OF MAX POOLING OPERATION ON AN INPUT CHANNEL.



SOURCE: (Vasilev; Slater, 2019)
NOTE: In the figure, the Max Pooling Operation is characterized by having 1 Input Channel, 1 Output Channel, 0-pixel padding, 2-pixel stride, and 2x2 pixel Window Size.

FIGURE 7 – EXAMPLE OF AVERAGE POOLING OPERATION ON AN INPUT CHANNEL.



SOURCE: (Vasilev; Slater, 2019)
NOTE: In the figure, the Average Pooling Operation is characterized by having 1 Input Channel, 1 Output Channel, 0-pixel padding, 2-pixel stride, and 2x2 pixel Window Size.

and contribute to tasks like lane detection and traffic sign recognition. By analyzing images and enabling depth estimation, CNNs enhance the vehicle's perception, ensuring safe navigation. These networks also play a crucial role in sensor fusion, combining data from various sources for a comprehensive understanding of the driving environment. CNNs are instrumental in empowering AVs with the necessary skills for effective and safe navigation.

2.2 OPTIMIZATION OF DEEP LEARNING NEURAL NETWORKS

Optimizing DL models involves improving various components to achieve better performance and efficiency. This includes choosing suitable architectures, fine-tuning hyperparameters, handling weight initialization, applying normalization, selecting AFs, optimizing learning rates, using enhanced data, exploring quantization and pruning techniques, considering regularization, leveraging parallelization, utilizing HAs, and adopting Transfer Learning (TL) (Lee et al., 2023). The iterative process of experimentation is crucial to finding the ideal combination and enhancing the capabilities of models for specific tasks.

2.2.1 Quantization

Quantization in DL involves reducing the precision of numerical representations, specifically weights and activation values, to enhance model efficiency when deploying on resource-constrained devices. Two main types of quantization are weight quantization, which focuses on reducing the precision of model parameters, and activation quantization, which targets intermediate activation values during inference.

In weight quantization, the process involves mapping learned weights using high-precision formats, such as 32-bit floating-point numbers, to lower-bit representations, such as 8-bit fixed-point or Posits. This minimizes memory requirements and computational costs. Training models with quantization in mind and employing techniques like fine-tuning help mitigate precision loss.

Activation quantization involves reducing the precision of activation values flowing through the NN during inference. Training phase statistics guide the quantization scheme, with techniques like fixed-point or dynamic quantization determining the precision. Fine-tuning and training with quantization in mind can also be applied before post-quantization to recover accuracy.

In general, quantization is a fundamental optimization strategy, balancing model efficiency and accuracy, enabling DL model deployment on various platforms with varying computational capabilities.

Quantization, which reduces precision in DL models, collaborates with hardware acceleration for optimized deployment. Lower precision arithmetic benefits specialized hardware such as GPUs , TPUs , FPGAs, and microcontrollers, enhancing efficiency and enabling real-time inference on edge devices. Challenges include compatibility and precision trade-offs, addressed through fine-tuning and dynamic quantization. In summary, this integration optimizes performance and energy efficiency across a range of devices, from edge devices to datacenters.

2.3 CENTERFUSION

CenterFusion (Nabati; Qi, 2021) is an algorithm designed to enhance the 3D object detection and classification in AVs through effective sensor fusion. It addresses the challenges of combining radar and camera data by leveraging their complementarities. It does this through a middle-fusion strategy to accurately associate radar detections with the central points of objects in camera images, overcoming challenges of spatial misalignment and object occlusion.

The algorithm uses CNNs as fundamental components in its architecture for 3D object detection and classification in AVs. The key role of CNNs in CenterFusion is to extract essential features from both radar and camera data.

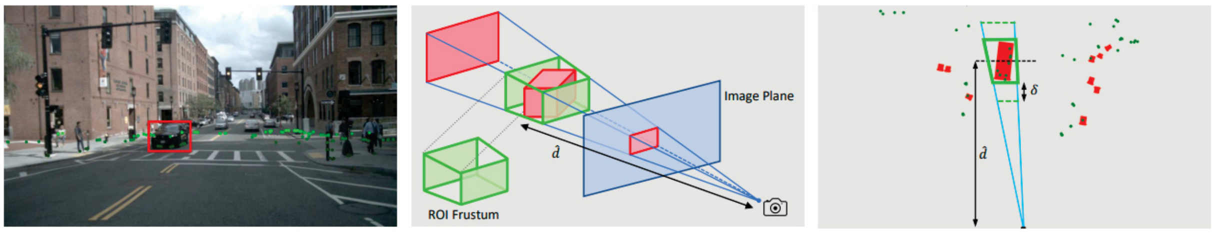
To achieve this, CenterFusion adopts CenterNet (Zhou et al., 2019), an approach that utilizes a central point detection network for the preliminary detection of objects in the image. For the camera data, a modified version of a Deep Layer Aggregation (DLA) NN (Yu et al., 2018) serves as the backbone for the process of 3D object detection and classification. The DLA is applied to the input images, generating FMs that capture relevant information about objects in the scene. The main regression heads of this CNN then use these features to predict object centroids, 2D size, 3D dimensions, depth, rotation, and other properties.

To handle inaccuracies in radar height information, CenterFusion introduces a preprocessing step called Pillar Expansion, providing a more accurate representation of the physical objects detected by radar. The CNN plays a crucial role in processing these expanded radar representations, which are then fused with image features to enhance the accuracy of 3D object detection and classification.

Radar features are generated by associating the expanded radar representations with objects in the image using a frustum-based method that leverages the properties of objects initially estimated by the primary regression heads. This frustum association mechanism introduces a method that uses the object's 2D bounding box, its estimated depth, and its size to form a 3D Region of Interest (RoI) frustum. This process, illustrated in FIGURE 8, significantly reduces the number of radar detections that require association by ignoring points outside the frustum. During training, the actual 3D bounding box of the object is used for more accurate association. Conversely, in testing, the RoI frustum is generated using estimated 3D bounding boxes, adjusted with the δ parameter to accommodate inaccuracies in depth estimation. This approach helps deal with overlapping objects, ensuring separate RoI frustums for distinct objects and resolving multiple detection association issues by selecting the nearest radar detection within the RoI frustum (Nabati; Qi, 2021).

Next, to incorporate radar data, the algorithm is extended with secondary

FIGURE 8 – CENTERFUSION'S FRUSTUM ASSOCIATION MECHANISM.

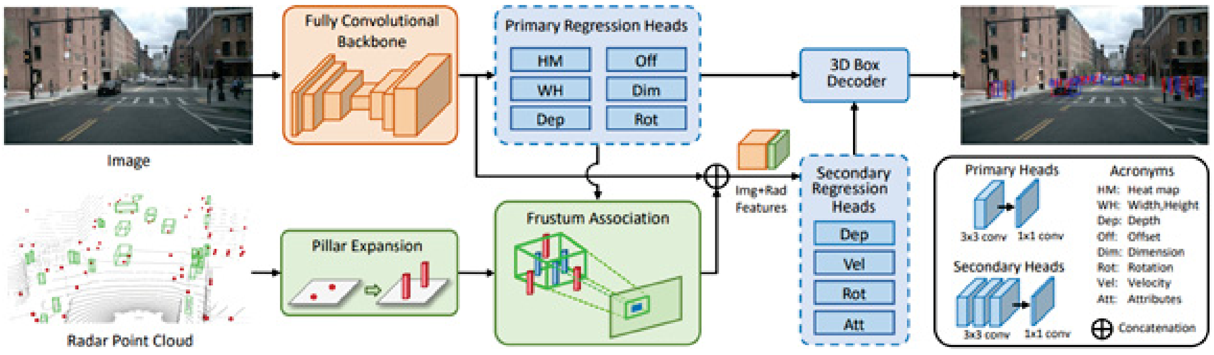


SOURCE: Nabati e Qi (2021)

NOTE: In the left image, an object is detected using image features. In the middle, the RoI frustum is generated based on the object's 3D bounding box. On the right, the top view of the RoI shows radar detections within the frustum. The δ parameter increases the frustum size, while $\hat{d}$ is the actual depth during training and the estimated one during testing.

regression heads dedicated to processing radar and camera-based features. The CNN is then trained using this fusion of radar and image features, enabling it to learn joint representations and effectively combine both sensing modalities, enhancing the initial estimation of crucial object properties such as depth, speed, rotation, and attributes. The overall architecture of CenterFusion can be visualized in FIGURE 9.

FIGURE 9 – OVERALL ARCHITECTURE OF CENTERFUSION.



SOURCE: Nabati e Qi (2021)

CenterFusion undergoes a comprehensive evaluation using the NuScenes dataset, outperforming existing camera-only based algorithms and demonstrating significant improvements in the NDS and speed estimation accuracy. Comparisons with other approaches, including camera-based models like CenterNet and MonoDIS, as well as a Light Detection and Ranging (LIDAR)-based method (InfoFocus), highlight CenterFusion's superiority in real-world scenarios (Nabati; Qi, 2021).

The overall architecture of CenterFusion demonstrates the integration of CNNs as a useful tool for feature extraction and 3D object detection and classification, showing its adaptability to handle various sensor data and achieve robust 3D object detection in real-world scenarios.

2.4 EVALUATION METRICS FOR OBJECT DETECTION AND CLASSIFICATION

In the context of object detection and classification, various metrics are used to evaluate the performance of algorithms. These metrics are essential for understanding how well a model can detect and classify objects in space. Among these metrics, those corresponding to the evaluation of the NuScenes dataset are included, such as:

- Average Precision (AP): This is a commonly used metric in object detection and classification. It measures the average precision at different levels of recall. Precision refers to the proportion of true positive detections among all positive detections, while recall is the proportion of true positive detections among all actual positives. AP is often calculated for each class separately and then averaged across classes. Thus,

$$AP = \frac{\sum_{k=1}^{n}(Recall_k - Recall_{k-1}) \cdot Precision_k}{n} \tag{2.8}$$

  Where $Precision_k$ and $Recall_k$ are the precision and recall at the k^{th} threshold. n is the number of thresholds.

- Average Translation Error (ATE): This metric is used to evaluate the accuracy of the detected position of an object and measures the average Euclidean distance between the centers of predicted and true bounding boxes. Lower values indicate better performance. Thus,

$$ATE = \frac{1}{N}\sum_{i=1}^{N}\sqrt{(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2} \tag{2.9}$$

  Where, (x_i, y_i) are the true coordinates of the object's center, and $(\hat{x}_i, \hat{y}_i)$ are the predicted coordinates. N is the number of samples.

- Average Scale Error (ASE): This metric evaluates how well the predicted bounding box size matches the actual object size. Thus,

$$ASE = 1 - IOU \tag{2.10}$$

  Where IOU is calculated as the ratio of the intersection area to the union area of the estimated and true bounding boxes after aligning their centers and orientations. This metric of Intersection over Union (IOU) quantifies the overlap between estimated and true bounding boxes and is given by

$$IOU = \frac{a_{gt} \cap a_e}{a_{gt} \cup a_e} \tag{2.11}$$

  In the expression, a_{gt} and a_e are the areas of the true and estimated bounding boxes, respectively. The IOU can range from 0 to 1, indicating the degree of overlap of the bounding boxes: 0 for no overlap, 1 for perfect alignment.

- Average Orientation Error (AOE): This measures the accuracy of the predicted orientation of detected objects. It is often calculated as the average angular difference between the orientations of the predicted and true bounding boxes. Thus,

$$AOE = \frac{1}{N}\sum_{i=1}^{N} \min(\Delta\theta_i, 2\pi - \Delta\theta_i) \tag{2.12}$$

  $\Delta\theta_i$ is the difference in the predicted and actual orientation angle for each instance.

- Average Velocity Error (AVE): Relevant in scenarios where objects are in motion (such as in AV applications), this metric measures the accuracy of the predicted velocity of detected objects. Thus,

$$AVE = \frac{1}{N}\sum_{i=1}^{N} |v_i - \hat{v}_i| \tag{2.13}$$

  Where v_i is the true velocity, and $\hat{v}_i$ is the predicted velocity.

- Average Attribute Error (AAE): This metric evaluates the accuracy of attribute predictions, such as the type of an object (e.g., car stopped or moving, pedestrian sitting or walking). It is typically used when the detection task involves classifying objects into multiple categories based on attributes. Thus,

$$AAE = 1 - Acc \tag{2.14}$$

  Where Acc is the attribute classification accuracy (each class may have different associated attributes that must also be predicted).

- NuScenes Detection Score (NDS): Specific to the evaluation of the NuScenes Dataset, the NDS is a composite metric that combines several of the above metrics (such as ATE, ASE, AOE) along with others like detection precision and recall. It provides a measure of a model's performance in object detection and classification tasks, particularly in the context of AD. Thus,

$$NDS = 0.1 \cdot \left(5 \cdot mAP + \sum_{mAE \in \{mATE, mASE, mAOE, mAVE, mAAE\}} \max(1 - mAE, 0)\right) \tag{2.15}$$

  Where mAP is the mean AP across all possible object classes, given by Equation 2.16.

$$mAP = \frac{1}{c}\sum_{i=1}^{c} AP_i \tag{2.16}$$

  Where c is the total number of classes. Metrics such as $mATE$, $mASE$, $mAOE$, $mAVE$, and $mAAE$ are defined in a similar way by using their associated average errors.

2.5 HARDWARE ACCELERATION

The optimization of specific tasks through HAs involves transferring these activities to dedicated components, resulting in significant increases in performance and energy efficiency. This approach encompasses a variety of applications, from using GPUs for graphics processing to TPUs for machine learning. FPGAs offer customization, ASICs handle specific functions, creating a scenario in which HAs simplify computing across various domains. In essence, different accelerators are designed to meet specific needs, improving efficiency in a wide range of applications.

In the realm of DL, HAs play a crucial role in optimizing computational operations during the training and inference of DNNs. GPUs and TPUs offer high parallel processing capacity and specialization in Artificial Intelligence (AI), while FPGAs provide flexibility and ASICs high energy efficiency. These solutions result in notable improvements in speed and efficiency, driving advances in complex tasks such as Natural Language Processing (NLP) and pattern recognition (Goodfellow et al., 2016).

2.5.1 Field-Programmable Gate Arrays

FPGAs represent a versatile and customizable solution in the realm of DL acceleration. These programmable integrated circuits provide a combination of resources that make them suitable for specific applications within the DL domain.

One of the characteristics of FPGAs is their customizable architecture, composed of programmable logic blocks and interconnections. This adaptability allows users to design and implement custom hardware circuits through Hardware Description Language (HDL), such as Very High-Speed Integrated Circuit Hardware Description Language (VHDL), or High-Level Synthesis (HLS), tailoring the hardware to the specific requirements of NN operations. This ability to create custom architectures is particularly advantageous in scenarios where optimized hardware is crucial.

FPGAs excel in parallel processing, facilitating the simultaneous execution of multiple operations. DL tasks, which frequently involve matrix multiplications and convolutions, can be efficiently parallelized on FPGAs. Additionally, the pipelining capability of FPGAs further enhances their throughput, making them adept at handling the parallel nature of NN computations.

Low latency and real-time processing are strengths of FPGAs. The ability to implement specific operations directly in hardware contributes to their suitability for applications requiring rapid decision-making, such as AVs or real-time analysis.

Energy efficiency is another notable feature of FPGAs, although less so than an ASIC. By enabling the customization of hardware for specific tasks, FPGAs avoid the overhead associated with executing unnecessary instructions, resulting in improved energy efficiency compared to general-purpose processors. However, while they allow adaptability to different architectures and NN models, the programming process can be more complex compared to traditional GPUs.

FPGAs find application in cases where their unique combination of features is advantageous, such as in edge computing, Internet of Things (IoT) devices, and embedded systems. Some systems adopt a hybrid approach, combining FPGAs with other accelerators to leverage the advantages of each type of hardware for different aspects of DL-related workloads.

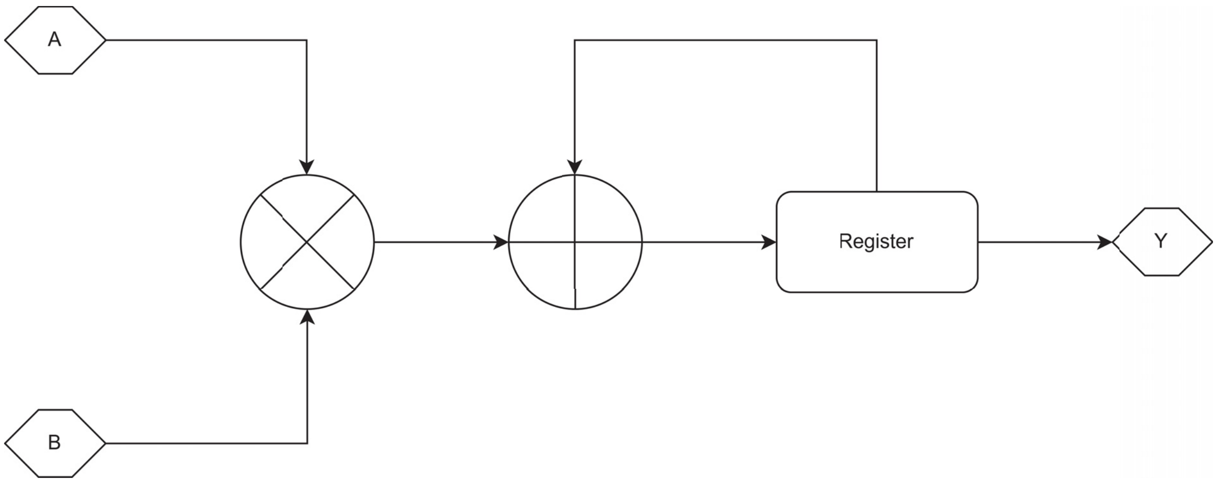
Finally, FPGAs offer a convenient solution for specific DL applications that demand customization, low latency, and energy efficiency.

2.5.2 Multiply-Accumulate Units (MAC Units)

A Multiply-Accumulate (MAC) Unit is defined as a computational entity with the inherent capability to properly execute both multiplication and accumulation operations. In its fundamental operation, this unit receives two input values, usually labeled as A and B, and proceeds to calculate their product $A \cdot B$, subsequently incorporating this product into an accumulation register, continuously updating its stored value. MAC Units are highly relevant across a wide range of domains. These include, but are not limited to, Digital Signal Processing (DSP), encompassing critical tasks such as filtering and convolution, as well as supporting scientific computing, where complex numerical problems, including differential equations and matrix multiplications, require its functionality. Additionally, their influence extends to ML, particularly in the training and inference of NNs, and is also crucial in graphics processing for rendering high-quality visual content. Furthermore, MAC Units are essential in the field of cryptography, where they enhance the performance and security of operations requiring complex mathematical calculations, including modular arithmetic and exponentiation.

In the field of DL, MAC Units serve as the backbone of NN operations. Their importance lies in their ability to efficiently perform multiply-accumulate operations, which are the central mathematical processes in NN training and inference. MAC Units excel at performing matrix multiplications, enabling the calculation of weighted sums and activations throughout the network. During forward propagation in inference and backpropagation in training, MAC Units play a fundamental role. They calculate the weighted sums of inputs and weights, followed by AFs, and are instrumental in calculating gradients to adjust network parameters. Such behavior is modeled in a simplified manner in the diagram of FIGURE 10, for a generic MAC Unit corresponding to a Processing Element (PE) of a DNN. In CNNs, MAC Units are specially adapted for convolution operations. This involves the element-wise multiplication and accumulation of filter weights by input data, allowing CNNs to extract features from images. Given the computational demands of DL, specialized hardware such as GPUs and TPUs incorporate dedicated MAC Units designed for efficient parallel processing. These HAs drastically improve the performance of DL applications.

FIGURE 10 – DIAGRAM OF A MULTIPLY-ACCUMULATE (MAC) UNIT.



SOURCE: The Author.

When designing a MAC Unit, optimizing efficiency and performance is crucial and often manifested through hardware enhancements, utilizing techniques such as pipelining, parallelism, and Single Instruction, Multiple Data (SIMD) methodologies (Hennessy; Patterson, 2017). Additionally, the choice of data format, whether fixed-point, floating-point, or Posit arithmetic, can be carefully adjusted to align with the specific application requirements. In scenarios where high processing capability is essential, such as in DL, adopting lower precision arithmetic like Posit numbers becomes an attractive strategy to accelerate computations, albeit with minimal final precision cost. Therefore, ongoing research and efforts focus on optimizing DL structures and models to maximize the benefits of MAC Units, with techniques like model pruning and quantization aimed at reducing computational complexity while preserving model accuracy.

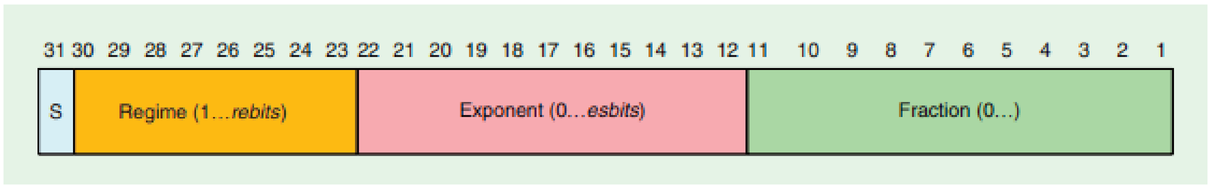
2.6 POSIT

The Posit is a numerical representation scheme aimed at improving efficiency in terms of precision, space, and speed compared to traditional floating-point representations, initially proposed by Gustafson e Yonemoto (2017). It also has a wider dynamic range when compared to standard floating-point and other quantization standards. Posit arithmetic is based on a unary representation, which means that all numbers are represented as powers of a common base, usually 2.

A real number represented in Posit is composed of (see FIGURE 11): a sign bit, s; a regime field, r, whose value is given by k; an exponent field, es, whose value is given by e; a fraction field, f, whose value is given by m.

The three-field structure of Posits - regime, exponent, and fraction - optimizes representation efficiency. The regime field encodes the position of the first 1 bit, contributing to a compact representation. The exponent field scales the number, similar to floating-point, and the fraction field contains the remaining bits, representing the fractional part.

FIGURE 11 – FIELDS OF POSIT NUMERICAL REPRESENTATION AND ARITHMETIC.



SOURCE: Cococcioni et al. (2021)

The function of the sign bit is to indicate whether the number represented is positive when $s = 0$ or negative otherwise. The regime field *r*, in turn, determines the range of values to which the number belongs. The regime of a Posit can be visualized as a shift applied to the range of possible exponents of a number. This shift, which occurs in steps of $2^{es\ bits}$, can go either left when $k < 0$ or right when $k > 0$. For a given k, and knowing that the exponent field has up to es bits, the exponent e_k, in base 2, of the represented number is given by equation 2.17

$$e_k = k \cdot 2^{es\ bits} + e \tag{2.17}$$

Where e is the value, in decimal, of the exponent field in the bit sequence representation, given by

$$e \in \{0, 1, 2 ... \ 2^{es\ bits} - 1\} \tag{2.18}$$

The value k of the regime r, in turn, is calculated by counting the number of consecutive 0 bits or 1 bits, after the sign bit, until the first 1 bit after the 0 bits or the first 0 bit after the 1 bits is found, not including them. If the bits to be counted are 0 bits (the bit after the sign bit, s, is a 0), then a countdown starts from -1, otherwise, a count-up starts from 0. Finally, the value of a real number represented in Posit is given by the expression

$$\begin{aligned} x &= (-1)^s \cdot 2^{e_k} \cdot (1 + \frac{m}{2^{f\ bits}}) \\ &= (-1)^s \cdot 2^{k \cdot 2^{es\ bits} + e} \cdot (1 + \frac{m}{2^{f\ bits}}) \\ x &= (-1)^s \cdot (2^{2^{es\ bits}})^k \cdot 2^e \cdot (1 + \frac{m}{2^{f\ bits}}) \end{aligned} \tag{2.19}$$

Some examples of mathematical constants represented using Posit (32, 2) can be seen in TABLE 1.

TABLE 1 – EXAMPLES OF MATHEMATICAL CONSTANTS IN POSIT (32, 2).

Constant	Value	s	r (k)	es (e)	f (m)
π	3.141593_{10}	0_2 (+)	10_2 (0_{10})	01_2 (1_{10})	$100100100001111110110101010_2$ $(76,610,986_{10})$
e	2.718282_{10}	0_2 (+)	10_2 (0_{10})	01_2 (1_{10})	$010110111111000010101000110_2$ $(48,203,078_{10})$

SOURCE: The Author.

The values of these constants can be obtained directly from the bit sequence representation using expression 2.19, as follows

$$x = (-1)^0 \cdot (2^{2^2})^0 \cdot 2^1 \cdot (1 + \frac{76.610.986}{2^{27}}) = 3,141593 \tag{2.20}$$

for π, and as follows

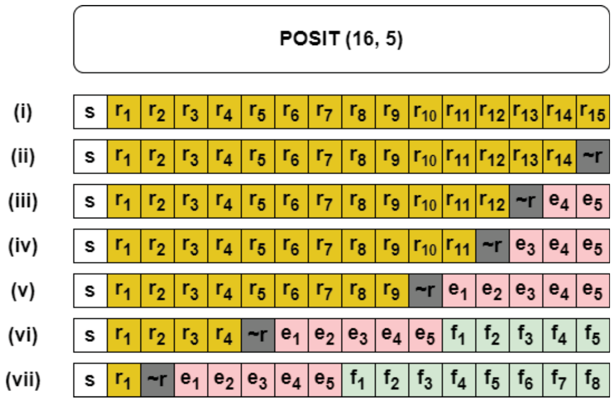
$$x = (-1)^0 \cdot (2^{2^2})^0 \cdot 2^1 \cdot (1 + \frac{48.203.078}{2^{27}}) = 2,718282 \tag{2.21}$$

for Euler's Number (e).

The Posit representation is a type of representation whose fields have variable sizes, distinguishing it also from the standard floating-point. A Posit is characterized only by its total size (width), in number of bits, and the maximum size, es bits, that its respective exponent field can have. Implicit 0 bits are assumed to the left of the exponent in cases where the number of bits in this field is smaller than es bits. In a Posit, the sizes of both the exponent and fraction fields depend on the size of the regime field, as can be seen in FIGURE 12.

However, this variable characteristic of Posits introduces a higher level of complexity when implementing their respective operators and arithmetic units in hardware. Nevertheless, the presence of the regime field in Posit representation is what gives it a wider dynamic range when compared to the standard floating-point arithmetic. With this,

FIGURE 12 – EXAMPLES OF POSIT (16, 5), WITH 16-BIT BIT WIDTH AND UP TO 5 EXPONENT BITS.


POSIT (16, 5)
(i) s r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 r15
(ii) s r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 r13 r14 ~r
(iii) s r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 r12 ~r e4 e5
(iv) s r1 r2 r3 r4 r5 r6 r7 r8 r9 r10 r11 ~r e3 e4 e5
(v) s r1 r2 r3 r4 r5 r6 r7 r8 r9 ~r e1 e2 e3 e4 e5
(vi) s r1 r2 r3 r4 ~r e1 e2 e3 e4 e5 f1 f2 f3 f4 f5
(vii) s r1 ~r e1 e2 e3 e4 e5 f1 f2 f3 f4 f5 f6 f7 f8

SOURCE: The Author.

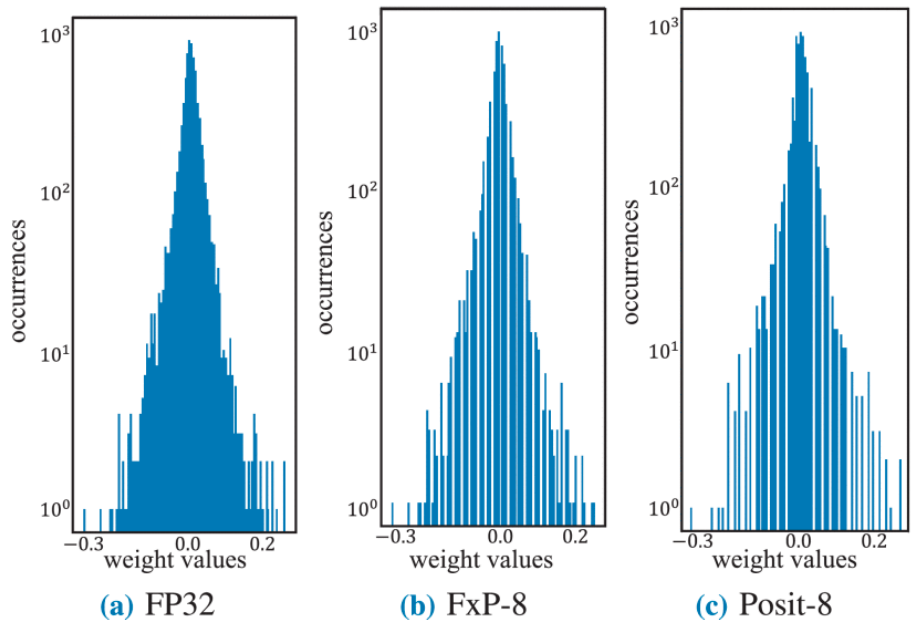
Posit arithmetic introduces a dynamic and adaptable numerical representation, adjusting precision based on the magnitude of the numbers. This contrasts with traditional fixed-point and floating-point systems, which maintain uniform precision.

The decoding of a Posit begins with determining the sign bit s, and if this bit is 1, the two's complement of the bit sequence that composes the Posit must be taken. Subsequently, using the resulting bit sequence, the value k of the regime is calculated as previously described. Next, the value, e, of the exponent, if present, is calculated. Finally, the value of the fraction, f, of the mantissa can be obtained if this field exists for this number.

As can be seen in FIGURE 13, for the statistical distribution of values commonly obtained for the parameters of VGG-16 type NNs, the Posit format shows greater capacity in representing these possible values compared to fixed-point quantization, for example. This is due to the fact that the density of Posit representation for numbers close to 0 is higher than the density obtained using fixed-point representations in the same range for the same bit width (Nambi et al., 2021). Due to this, quantizing normalized parameters of floating-point NNs using Posit results in significantly less degradation in the final accuracy of the algorithm when compared to other types of fixed-point quantization (Nambi et al., 2021).

A distinctive aspect of Posit arithmetic is the gradual handling of overflow and underflow. Instead of abrupt saturation, Posits make a smooth transition to infinity or zero, providing a more graceful degradation in extreme cases. This can be advantageous when a gradual loss of precision is more acceptable than sudden failures. The ability of Posits to satisfactorily handle extreme values is beneficial in DL applications, where certain operations involve exceptionally large or small numerical values. The gradual handling of overflow and underflow mitigates the risk of numerical instability. Additionally, the absence of a special Not-a-Number (NaN) value simplifies the handling

FIGURE 13 – WEIGHT DISTRIBUTION OF A VGG-16 NN FOR DIFFERENT NUMERICAL REPRESENTATIONS.



SOURCE: Nambi et al. (2021)
NOTE: (a) 32-bit Floating-Point; (b) 8-bit Fixed-Point; (c) 8-bit Posit.

of exceptional cases, contributing to more predictable behavior in numerical calculations.

Posit arithmetic aligns well with DL, offering potential advantages in terms of adaptability to dynamic ranges in neural activations and efficient use of HAs. The dynamic range of Posits aligns with the diverse magnitudes found in NN activations, allowing adaptive precision allocation. This adaptability is crucial in DL tasks, where input data and network parameters vary widely. Furthermore, the quire operation in Posit arithmetic enhances the efficiency of accumulation and summation of products, potentially resulting in faster and more accurate results. Faster and more accurate accumulation of products in Posit arithmetic is particularly relevant for optimization algorithms like gradient descent and operations involving matrix multiplication. This contributes to both faster convergence during model training and improved performance during inference.

In the context of HAs, the efficient handling of arithmetic operations by Posits can result in faster inference times and greater energy efficiency. Additionally, the variable precision of Posits allows for more efficient use of hardware resources, directing higher precision when needed and conserving resources in other cases.

However, for the practical integration of Posit arithmetic into DL frameworks, further research, standardization efforts, and industry consensus are required. Theoretical benefits need to translate into practical applications, considering factors such as compatibility with existing ecosystems and frameworks.

3 RELATED WORKS

3.1 SYSTEMATIC LITERATURE REVIEW

A Systematic Literature Review (SLR) involves defining a research question, conducting a systematic search for relevant studies, selecting, assessing the quality, synthesizing results, and transparently reporting the findings.

Initially, the research questions were defined as follows:

Q1) What models, datasets, and Posit quantization tools currently exist for NN conversion and inference emulation using the format;

Q2) What arithmetic operators and accelerators for Posit have been developed, as well as the tools used for such and the associated hardware platforms;

Next, the following search string was formulated to avoid limiting the search results to only a few dozen studies:

("Artificial Intelligence"OR "Machine Learning"OR "Deep Learning"OR "Neural Network") AND ("Posit Arithmetic"OR "Posit Encoding"OR "Posit Format"OR "Posit Number"OR "Posit Operator"OR "Posit Quantization"OR "Posit Representation")

Using this search string, 297 studies were retrieved from the indexing databases. 61 studies were selected for full analysis and data synthesis. The steps of the search for related works can be seen in FIGURE 14.

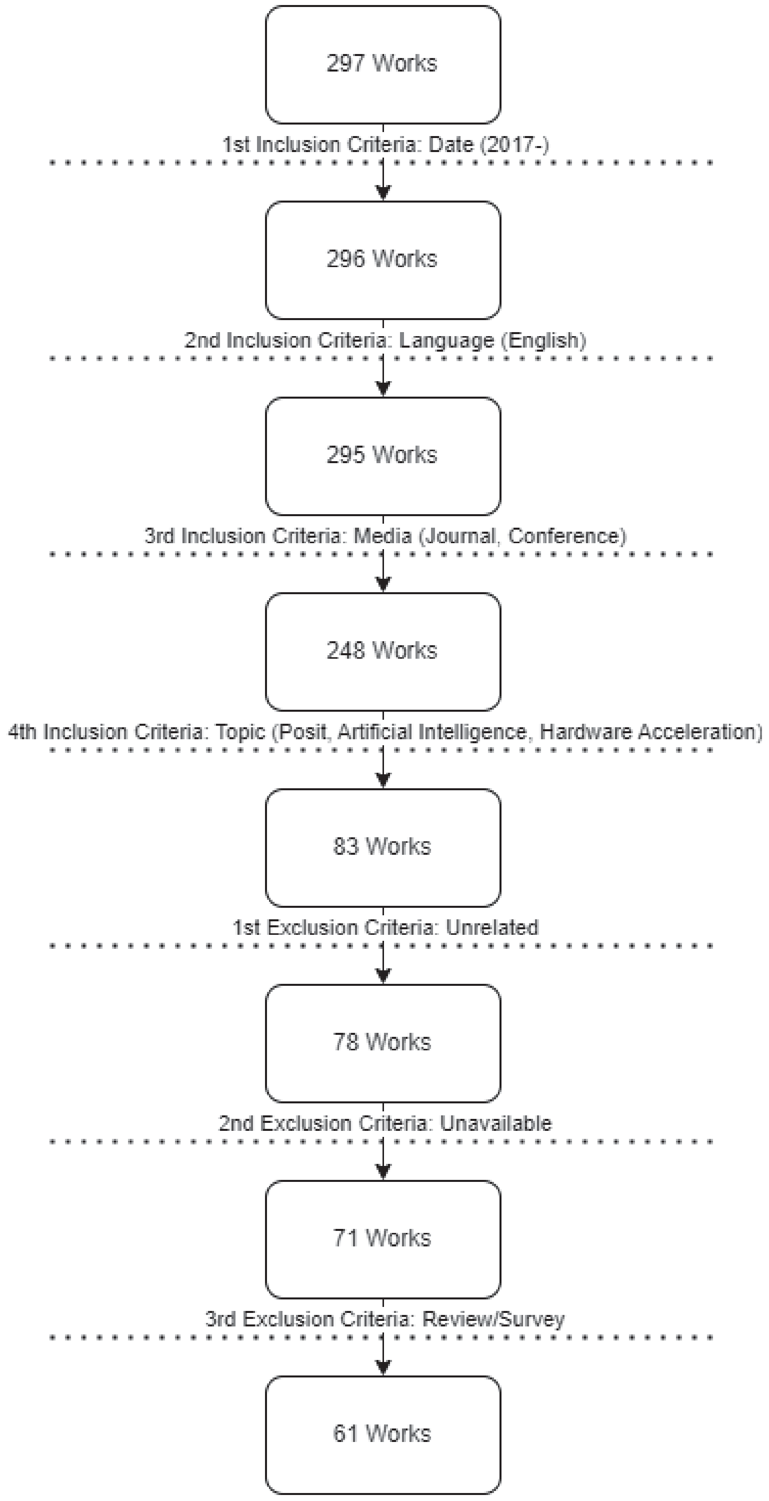
3.1.1 Literature Distribution

Upon completing the stages of the Systematic Literature Review, a diagram was created using VOSViewer software to visualize the distribution of literature across different authors and to identify co-authorship networks. This is represented in Figure FIGURE 15.

An analysis of the diagram shows that research on Posit numbers is still in its early development and is notably fragmented, suggesting substantial opportunities for further exploration into this emerging arithmetic approach, as proposed in this study. It's also interesting to note that since 2020, some of the most active authors in this field have slowed their research output, although there are notable exceptions.

One standout observation in the diagram is the trajectory of J. Gustafson, who originally introduced Posit numbers and arithmetic in Gustafson e Yonemoto (2017). However, his influence appears to have declined from 2020 onward. Conversely, there

FIGURE 14 – STEPS OF THE SLR.

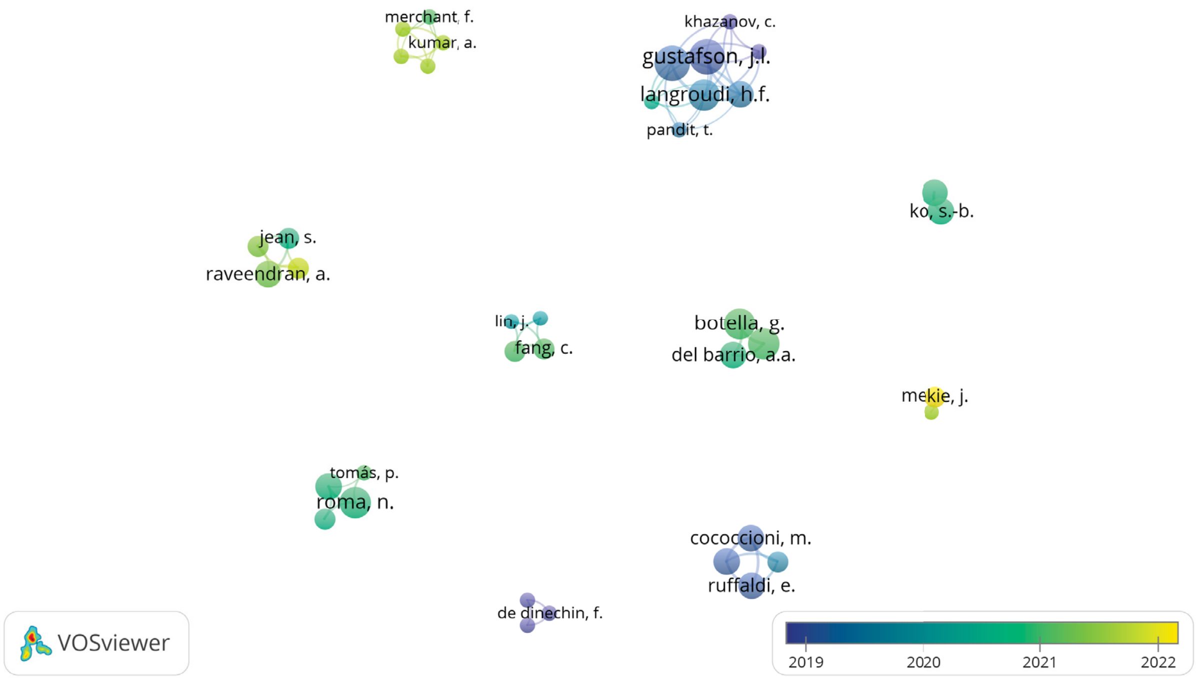

297 Works
1st Inclusion Criteria: Date (2017-)
296 Works
2nd Inclusion Criteria: Language (English)
295 Works
3rd Inclusion Criteria: Media (Journal, Conference)
248 Works
4th Inclusion Criteria: Topic (Posit, Artificial Intelligence, Hardware Acceleration)
83 Works
1st Exclusion Criteria: Unrelated
78 Works
2nd Exclusion Criteria: Unavailable
71 Works
3rd Exclusion Criteria: Review/Survey
61 Works

SOURCE: The Author.

has been a noticeable increase in contributions from other authors in recent years.

This landscape underscores the continued need for research and provides opportunities for new scholars to advance the field of Posit arithmetic and related numerical approaches.

FIGURE 15 – DISTRIBUTION OF LITERATURE BY AUTHOR / CO-AUTHOR.



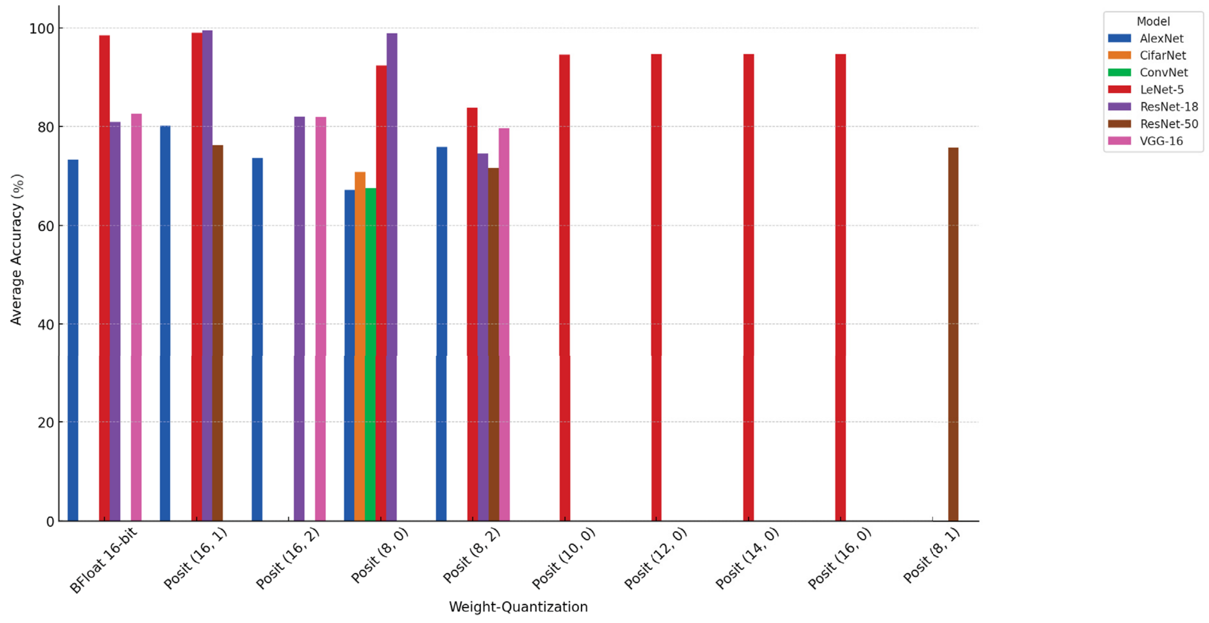
SOURCE: The Author.

3.2 DISCUSSION ON THE STATE-OF-THE-ART

The analysis of the State-of-the-Art (SOTA) on Posit numbers reveals an incipient panorama for this new numerical representation applied to DL. Firstly, there is a noticeable absence in the analyzed works of NN models specifically developed for AV applications. Most of the works prioritized the use of conventional networks, such as LeNet-5, ResNet-50/18, and AlexNet, evaluated on commonly used datasets for benchmarking these networks, mainly MNIST, Fashion MNIST, CIFAR-10, and ImageNet. These include the works of Cococcioni et al. (2020a,b), Edavoor et al. (2023), Glint et al. (2023), Gohil et al. (2021), Kumar e Gupta (2023), Langroudi et al. (2018), Langroudi et al. (2020), Lu et al. (2019), Murillo et al. (2020b, 2021, 2022a,b), Raposo et al. (2021), Walia et al. (2022), Wang et al. (2021), Zhang e Ko (2023) e Zolfagharinejad et al. (2022). An exception to this is the work of Zolfagharinejad et al. (2022), where the authors evaluate a Posit (8, 2) quantization on a ResNet-34 model, which relates to CenterFusion's DLA-34 backbone. Moreover, none of the methodologies exposed in previous works explicitly considered sensor fusion through DL, as exemplified by the CenterFusion method.

Nevertheless, from the analyzed works it is found that Posit numbers are indeed successful in DL applications, allowing for parameters' bit-width reduction with minimal degradation in the precision of the studied models. A summary of the average accuracy obtained after inference by the most recurring types of NN models in terms of the most frequent quantization formats in the analyzed works can be seen in FIGURE 16.

FIGURE 16 – AVERAGE ACCURACY X WEIGHT QUANTIZATION FORMAT.



SOURCE: The Author.

This was the case both when using the format solely for parameter storage in memory, with Posit-as-Storage (PaS), and when performing inference through emulation of arithmetic operations in Posit, with Posit-as-Arithmetic (PaA), either by executing models with hardware Posit operators emulated through software or by emulating Posit operations themselves using integer arithmetic or bitwise instructions running on a Central Processing Unit (CPU), exclusively, as in the case of the SoftPosit library (Leong, 2018). The Top-1 Accuracies obtained using both PaS and PaA strategies can be seen, respectively, in TABLE 2 and TABLE 3. However, there is a noticeable need for new tools that can support this new representation, facilitating its evaluation in more specific scenarios and diverse applications, both in software and hardware. Almost all the analyzed works propose custom methods, usually not openly disclosed or not comprehensible, for studying the format, which complicates the reproducibility of results and methodologies by other researchers. Moreover, there is a total absence of support from major software and hardware companies in their development tools for Posit numbers, whether in designing software models or implementing accelerators in FPGA or ASIC using Posit operators and PEs. Exceptions to this exist and are available online in open repositories, even though providing limited support for studying the representation, such as PositNN from Raposo et al. (2021), which was developed for training DNNs using Posit data formats with different precision levels. This open-source framework for C++, similar to PyTorch (Paszke et al., 2019), supports mixed precision and has been successful in experiments with different datasets, revealing performance improvements compared to traditional methods. Another significant contribution is presented by Murillo et al. (2020b) in an earlier study from 2020 where Deep PeNSieve is introduced, a framework that enables training and inference in DNNs using Posits. This Python framework stands out for allowing training and inference with Posits, as well as low-precision inference with 8-bit Posits through the use of fused operations. Deep PeNSieve was successfully evaluated on two NN architectures across different datasets, demonstrating flexibility and efficiency in handling Posits. Subsequently, Murillo et al. (2021) trained DNN models in Posit (32, 2) format using the Deep PeNSieve framework. After training, the models were quantized to Posit format (8, 0), and the research compared the inference results when standard and fused operations were used. In Nakahara et al. (2022), a library which integrates with PyTorch to optimize quire size for posit-based DNN models was developed. It replaces standard functions with posit-based ones, quantizes weights, and implements posit-based layers. Key operations, written in Rust for efficiency, handle conversion between floating-point and posit formats and perform arithmetic operations. The workflow involves converting models to posit-based operations and conducting inference, achieving high accuracy with reduced hardware area. This flexible and efficient library is valuable for optimizing posit-based DNNs in AI and edge computing.

TABLE 2 – TOP-1 ACCURACIES FOR MODEL AND DATASET COMBINATIONS FOR POSIT-AS-STORAGE (PAS).

Source	**Model**	***Dataset***	**Quantization**	**Accuracy (%)**
(Langroudi et al., 2018)	LeNet-5	MNIST	Posit (2, 0)	8.81
(Wang et al., 2021)	MobileNet-V2	CIFAR-10	Posit (8, 1)	94.20
(Lu et al., 2019)	ResNet-18	CIFAR-10	Posit ({8, 16}, [1, 2])	92.87
(Langroudi et al., 2020)	8-Layer CNN	Fashion MNIST	Adap. Posit (6, [0, 1])	92.48
(Langroudi et al., 2020)	ResNet-50	CIFAR-10	Adap. Posit (6, [0, 1])	72.56
(Langroudi et al., 2020)	11-Layer CNN	Fashion MNIST	Adap. Posit (5, [0, 1])	73.29
(Wang et al., 2021)	MobileNet-V2	ImageNet	Posit (8, 1)	71.20
(Lu et al., 2019)	ResNet-18	ImageNet	Posit (16, [1, 2])	71.09
(Langroudi et al., 2018)	ConvNet	CIFAR-10	Posit (8, 0)	67.54
(Langroudi et al., 2018)	AlexNet	ImageNet	Posit (8, 0)	55.07

SOURCE: The Author.

A trend observed from the analysis is the proposition of modifying the representation or using intermediate formats that facilitate the handling of Posit numbers, especially in developing more efficient hardware operators. Some examples include: Raw Posit (Zolfagharinejad et al., 2022), Generalized Posit (Langroudi et al., 2021), Adaptive Posit (Langroudi et al., 2020), Fixed Posit (Gohil et al., 2021), and Approximate Fixed Posit (Glint et al., 2023).

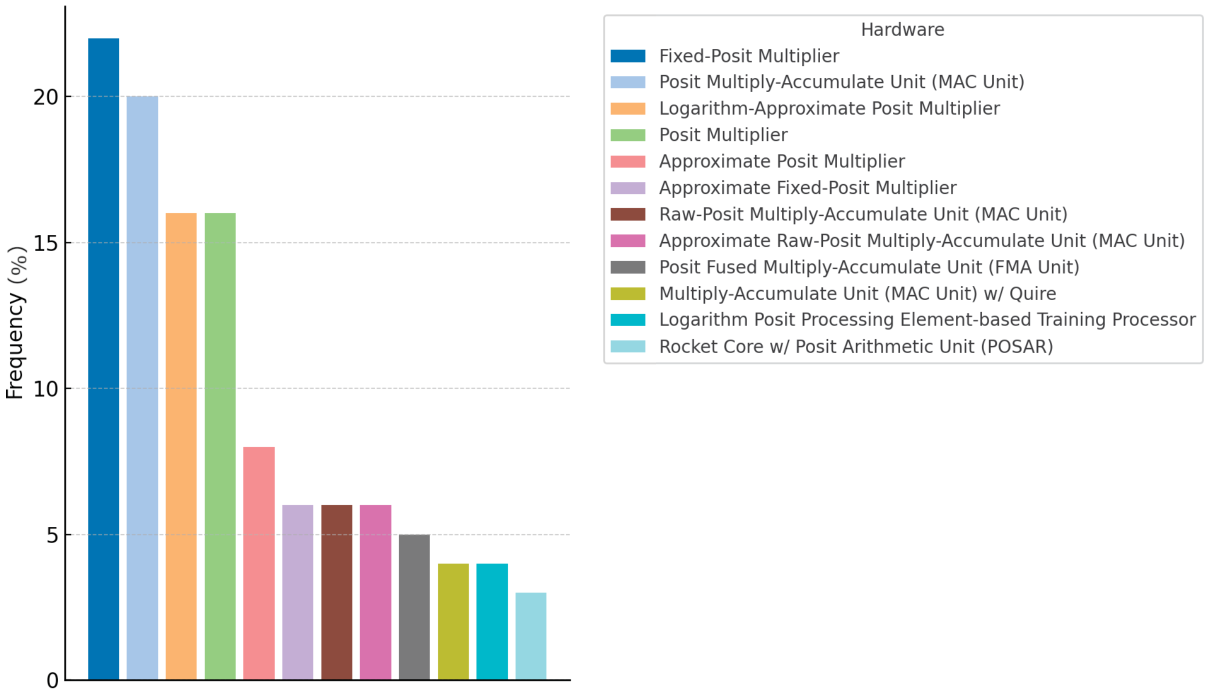
TABLE 3 – TOP-1 ACCURACIES FOR MODEL AND DATASET COMBINATIONS FOR POSIT-AS-ARITHMETIC (PAA).

Source	**Model**	***Dataset***	**Format**	**Accuracy (%)**
(Kumar; Gupta, 2023)	ResNet-18	MNIST	Posit (8, 0)	98.91
(Edavoor et al., 2023)	4-Layer CNN	Alphabet	Posit (8, 0)	97.00
(Langroudi et al., 2021)	EfficientNet-B0	CIFAR-10	Gen. Posit (6, [0, 2])	70.64
(Murillo et al., 2022b)	ResNet-18	SVHN	Posit (16, 2)	95.99
(Zhang; Ko, 2023)	MobileNetV2	CIFAR-10	Posit (16, 2)	93.65
(Walia et al., 2022)	3-Layer FCNN	MNIST	Posit (5, 2)	94.50
(Zhang; Ko, 2023)	ResNet-18	CIFAR-10	Posit (16, 2)	94.50
(Walia et al., 2022)	ISOLET	MNIST	Fixed-Posit (6, 3, 2)	94.16
(Cococcioni et al., 2020a)	LeNet-5	GTRSB	Posit (12, 0)	94.20
(Zhang; Ko, 2023)	VGG-16	CIFAR-10	Posit (16, 2)	94.06
(Immaneni et al., 2022)	784-100-64-10	MNIST	Posit (8, 4)	58.98
(Murillo et al., 2022a)	ResNet-50	CIFAR-10	Posit (16, 1)	89.87
(Nakahara et al., 2022)	ResNet-9	CIFAR-10	Posit (8, 1)	17.39
(Zhang; Ko, 2023)	AlexNet	CIFAR-10	Posit (16, 2)	91.70
(Raposo et al., 2021)	LeNet-5	FMNIST	Posit (8, 0)	13.84
(Murillo et al., 2021)	CifarNet	SVHN	Posit (8, 0)	89.13
(Murillo et al., 2022a)	Pre-ResNet-20	CIFAR-10	Posit (16, 1)	87.58
(Zolfagharinejad et al., 2022)	DPN92	CIFAR-10	Posit (8, 2)	86.64
(Zolfagharinejad et al., 2022)	ResNet-34	CIFAR-10	Posit (8, 2)	74.01
(Edavoor et al., 2023)	5-Layer CNN	CIFAR-10	Posit (32, 2)	85.00
(Langroudi et al., 2021)	EfficientNet-B4	ImageNet	Gen. Posit (8, [0, 2])	81.39
(Walia et al., 2022)	19-Layer CNN	CIFAR-10	Posit (8, 2)	83.43
(Murillo et al., 2022a)	Cuda-ConvNet	CIFAR-10	Posit (16, 1)	81.37
(Walia et al., 2022)	Xception	ImageNet	Fixed-Posit (10, 4, 2)	77.31
(Walia et al., 2022)	Inception-V3	ImageNet	Fixed-Posit (10, 4, 2)	76.09
(Tambe et al., 2020)	ResNet-50	ImageNet	Posit (16, 1)	76.10
(Murillo et al., 2022a)	ResNet-50	CIFAR-100	Posit (16, 1)	65.95
(Walia et al., 2022)	VGG-16	ImageNet	Fixed-Posit (8, 3, 2)	68.08
(Zhang; Ko, 2023)	MobileNetV2	ImageNet	Posit (16, 2)	70.52
(Walia et al., 2022)	VGG-19	ImageNet	Posit (8, 2)	70.92
(Gohil et al., 2021)	ResNet-18	ImageNet	Fixed-Posit (18, 6, 2)	70.26
(Raposo et al., 2021)	CifarNet	CIFAR-10	Mix. Posit (8, 2)	68.65
(Ciocirlan et al., 2021)	14-Layer CNN	CIFAR-10	Posit (16, 2)	68.15
(Zolfagharinejad et al., 2022)	LeNet-5	CIFAR-10	Posit (8, 2)	62.53
(Murillo et al., 2022a)	Pre-ResNet-20	CIFAR-100	Posit (16, 1)	59.96
(Zhang; Ko, 2023)	AlexNet	ImageNet	Posit (16, 2)	56.81
(Murillo et al., 2022a)	Cuda-ConvNet	CIFAR-100	Posit (16, 1)	52.36
(Cococcioni et al., 2020b)	LeNet-5	MNIST	Posit (16, 0)	99.10

SOURCE: The Author.

Regarding the development of hardware support for the Posit format, there is a predominance of multiplication operators and MAC Units in the analyzed works, as it can be seen in FIGURE 17. This is because multiplication operators are the most resource-intensive in terms of hardware and latency. Nonetheless, these are the most critical elements that significantly impact the performance of matrix multiplication operations — fundamental in the inference and training of DL models — and consequently, MAC Units. This justifies the focus of several works exclusively on optimizing Posit multiplication operators, such as Essam et al. (2022), Glint et al. (2023), Gohil et al. (2021), Immaneni et al. (2022), Kant e Thakur (2021), Murillo et al. (2020a, 2022a), Norris e Kim (2021), Shekhawat et al. (2021), Uguen et al. (2019), Walia et al. (2022) e Zhang e Ko (2020, 2021, 2023).

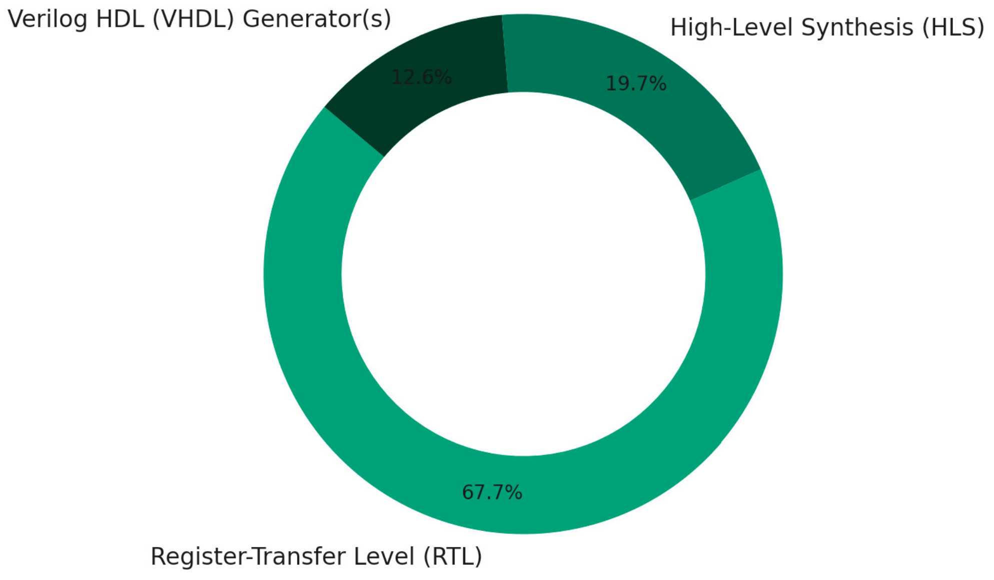
FIGURE 17 – DISTRIBUTION OF HARDWARE DISTRIBUTION FOR POSIT INFERENCE.



SOURCE: The Author.

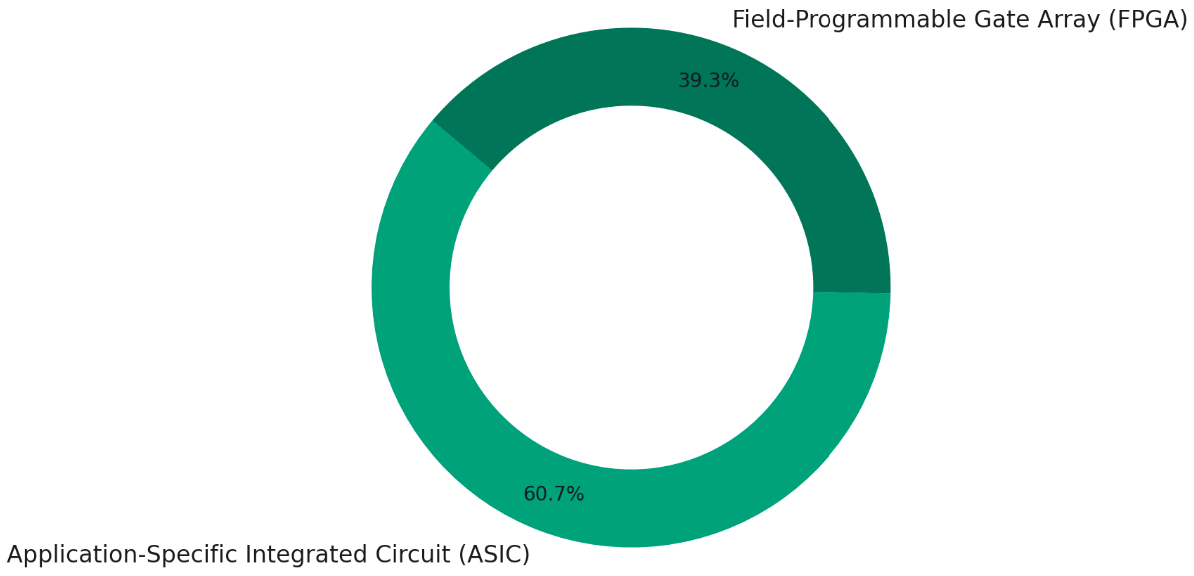
For designing operators and PEs, a significant portion of the works utilized HDLs at Register-Transfer Level (RTL), mainly VHDL, followed by HLS, as shown in FIGURE 18, with the most common types of hardware platforms given in FIGURE 19. It is interesting to note the presence of parameterized VHDL code generators in the works, allowing the generation of arithmetic operators and MAC Units with various bit-width, exponent field sizes, among other parameters. Podobas e Matsuoka (2018) introduces POSGEN, a tool for creating custom Posit hardware operators, generating synthesizable VHDL descriptions and facilitating FPGA integration into OpenCL flows. Jaiswal e So (2019) presents PACoGen, an open-source hardware generator for customizable Posit arithmetic operations, optimized for FPGA and ASIC platforms. Zhang et al. (2019) describes a C-based generator for Posit MAC Units, producing and testing VHDL code for various configurations, emphasizing its use in deep learning applications. Murillo et al. (2020a) discusses parameterized Posit arithmetic units in the FloPoCo framework, showcasing significant efficiency improvements. Murillo et al. (2021) and Murillo et al. (2022a) further use FloPoCo to design and verify Posit MAC Units, ensuring accuracy through extensive testing and synthesis. This corroborates the need for tools that facilitate the study of the format, allowing iterative study and design space exploration for Posit numbers, especially when developing HAs for specific DL models, each requiring different values for the representation parameters.

FIGURE 18 – DISTRIBUTION OF DESIGN LEVELS FOR POSIT HARDWARE.



SOURCE: The Author.

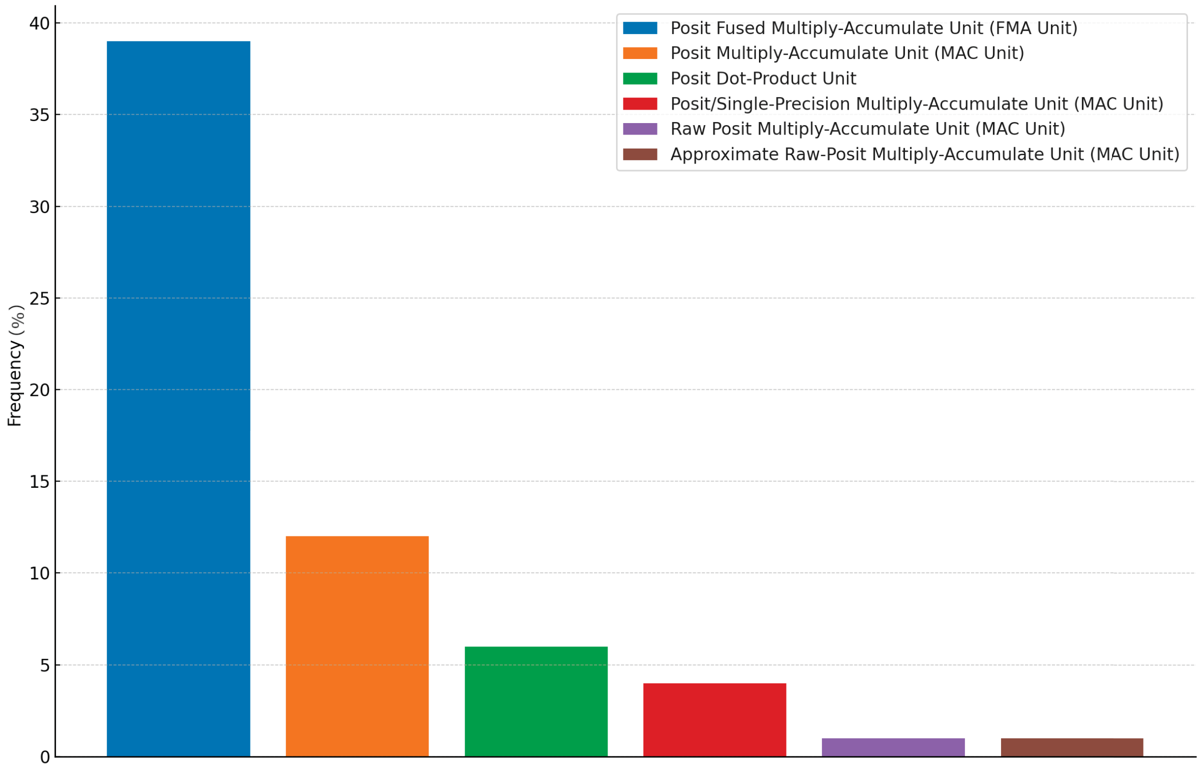
FIGURE 19 – DISTRIBUTION OF PLATFORM TYPES FOR POSIT HARDWARE.



SOURCE: The Author.

FIGURE 20 provides information about the distribution of specific categories related to Posit PEs within the analyzed works. It reveals a prominent presence of Posit Fused Multiply-Accumulate (FMA) Units as the most frequent category with the second most common category being Posit Multiply-Accumulate Units (MAC Units). FMA units and MAC Units differ mainly in precision and performance. MAC Units perform multiplication followed by addition as two separate steps, which can introduce rounding errors and result in higher latency. In contrast, FMA units combine multiplication and addition into a single operation with full precision, rounding only the final result. This reduces rounding errors and improves accuracy. The presence of Posit Dot-Product Units, although representing a smaller fraction, indicates the development of specialized PEs for vector and matrix calculations. Interestingly, the occurrences also include Hybrid Posit/Single-Precision MAC Units (Crespo et al., 2022), suggesting a hybrid approach that supports both Posit and traditional floating-point formats. This flexibility is advantageous in transition technologies or in applications that require multiple number formats. The presence of Raw Posit MAC Units and Approximate Raw Posit MAC Units, though less frequent, indicate niche or specialized applications where raw or approximate Posit calculations are preferred, to prioritize speed or energy efficiency over precision (Zolfagharinejad et al., 2022).

FIGURE 20 – TOP-6 CATEGORIES OF PROCESSING ELEMENTS IN POSIT.



SOURCE: The Author.

TABLE 4, TABLE 5, and TABLE 6 synthesize the occurrences corresponding to MAC Units in terms of delay, area, and power, respectively, filtered by the lowest delay, area, and power, by PE category.

Accelerators for Posit have also been developed. Wang et al. (2021) introduces the Logarithm Posit Processing Element (LPE) accelerator, using the Posit

TABLE 4 – TOP-1 DELAY BY PROCESSING ELEMENT CATEGORY.

Source	Category	Format	Delay (ns)
(Mishra et al., 2022)	BFloat FMA Unit	Bfloat 16-bit	108.979
(Mishra et al., 2022)	Fixed-Point FMA Unit	Q6.10	88.113
(Zolfagharinejad et al., 2022)	Fixed-Point MAC Unit	Fixed-Point 8-bit	1.210
(Mishra et al., 2022)	Floating-Point FMA Unit	Single-Precision	211.572
(Li et al., 2023)	Posit Dot-Product Unit	Posit (13, 2)	1.600
(Crespo et al., 2022)	Posit FMA Unit	Posit (8, 2)	0.650
(Nakahara et al., 2022)	Posit MAC Unit	Posit (8, 1)	1.920
(Zolfagharinejad et al., 2022)	Raw Posit MAC Unit	Raw-Posit (11, 2)	0.600
(Crespo et al., 2022)	Unified Posit/IEEE 754 VMA Unit	16-bit	1.500

SOURCE: The Author.

TABLE 5 – TOP-1 AREA BY PROCESSING ELEMENT CATEGORY.

Source	Category	Format	Area (um2)
(Mishra et al., 2022)	BFloat FMA Unit	Bfloat 16-bit	113.038
(Mishra et al., 2022)	Fixed-Point FMA Unit	Q6.10	266.905
(Zolfagharinejad et al., 2022)	Fixed-Point MAC Unit	Fixed-Point 8-bit	1278.000
(Mishra et al., 2022)	Floating-Point FMA Unit	Single-Precision	359.736
(Li et al., 2023)	Posit Dot-Product Unit	Posit (13, 2)	7694.820
(Mishra et al., 2022)	Posit FMA Unit	Posit (16, 1)	115.721
(Nakahara et al., 2022)	Posit MAC Unit	Posit (8, 1)	509.380
(Lu et al., 2019)	Posit/Single-Precision MAC Unit	Posit (8, 2)	1032.000
(Zolfagharinejad et al., 2022)	Raw Posit MAC Unit	Raw-Posit (11, 2)	1481.000
(Crespo et al., 2022)	Unified Posit/IEEE 754 VMA Unit	16-bit	51563.000

SOURCE: The Author.

TABLE 6 – TOP-1 POWER BY PROCESSING ELEMENT CATEGORY.

Source	Category	Format	Power (mW)
(Mishra et al., 2022)	BFloat FMA Unit	Bfloat 16-bit	97.452
(Mishra et al., 2022)	Fixed-Point FMA Unit	Q6.10	211.912
(Zolfagharinejad et al., 2022)	Fixed-Point MAC Unit	Fixed-Point 8-bit	0.15
(Mishra et al., 2022)	Floating-Point FMA Unit	Single-Precision	285.866
(Li et al., 2023)	Posit Dot-Product Unit	Posit (13, 2)	3.66
(Murillo et al., 2021)	Posit FMA Unit	Posit (16, 2)	10.02
(Lu et al., 2019)	Posit/Single-Precision MAC Unit	Posit (8, 2)	0.35
(Zolfagharinejad et al., 2022)	Raw Posit MAC Unit	Raw-Posit (11, 2)	0.073
(Crespo et al., 2022)	Unified Posit/IEEE 754 VMA Unit	16-bit	99

SOURCE: The Author.

format, logarithmic domain calculations, and a two-stage floating-point MAC Unit to enhance energy efficiency. Implemented with a 28 nm Complementary Metal-Oxide-Semiconductor (CMOS) process, LPE reduces power consumption and area while maintaining the same dynamic range. Similarly, Neves et al. (2020) presents the Reconfigurable Tensor Unit (RTU) with Variable Precision Vector Multiply-Accumulate (VMA) Units, a reconfiguration mechanism for vector precisions, and an automatic data streaming infrastructure. TABLE 7 and TABLE 8 summarize these accelerators in terms of area and power.

TABLE 7 – TOP-1 AREA BY ACCELERATOR CATEGORY.

Source	Category	Format	Area (um2)
(Wang et al., 2021)	Logarithm Posit PE (LPE)	Posit (8, 1)	5280000
(Neves et al., 2020)	Reconfigurable Tensor Unit (RTU)	Posit (16, 1)	14204000

SOURCE: The Author.

TABLE 8 – TOP-1 POWER BY ACCELERATOR CATEGORY.

Source	Category	Format	Power (mW)
(Wang et al., 2021)	Logarithm Posit PE (LPE)	Posit (8, 1)	11-343
(Neves et al., 2020)	Reconfigurable Tensor Unit (RTU)	Posit (16, 1)	11708

SOURCE: The Author.

Finally, there is a significant challenge in studying Posit numbers, mainly due to the difficulty of implementing them in specific models and applications. Although there are movements towards standardizing the format, the absence of robust tools that provide generalized support for Posits in the DL domain, from design to implementation, significantly hinders the study of the format and its practical application. Such tools would allow for a fairer and more transparent comparison of methodologies and results, and also promote the standardization of the most relevant metrics in the Posit representation study field.

It is from this realization that this work is sustained, attempting to promote more general and transparent tools for the use of Posit numbers in NN models, in conjunction with well-established tools in the DL domain, and demonstrate their use in a specific and unexplored context. The aim is to not reinvent the wheel as much as possible, exploring the tools proposed in other works, modifying them if necessary in an intelligible way so that other researchers and future works can use the same tools and follow a similar methodology, continually improving them until there is a robust set of methodologies and tools for the facilitated, comprehensive, and non-exclusive study of this numerical format, both in software and hardware.

4 METHODOLOGY

As previously mentioned, CenterFusion uses a middle-fusion approach for fusing radar and camera data. The development of CenterFusion was carried out using the PyTorch framework, which offers a quantization API. Leveraging PyTorch's Quantization API, custom quantization capabilities were explored

Specifically, a custom quantization and inference framework, Posits4Torch, was developed. Posits4Torch is based on Deep PeNSieve (Murillo et al., 2020b), on SoftPosit (Leong, 2018), and on Python Productivity for ZYNQ (PYNQ) (AMD, 2016). Deep PeNSieve, a DL framework based on SoftPosit, is designed for training and inference of NNs using Posit numbers and arithmetic. It stands out for its compatibility with the TensorFlow framework but is also based on the NumPy library, which allowed its integration with Posits4Torch. Nonetheless, Posits4Torch was later extended to support FPGA-accelerated Posit inference through an auxiliary package - Posits4TorcHA - which features a custom hardware accelerator.

CenterFusion's Posit quantization was achieved through the application of Posits4Torch / Posits4TorcHA. To evaluate the impact of this quantization, a comprehensive comparison was made between the Posit-quantized CenterFusion and its non-quantized counterpart. The evaluation mainly covered CenterFusion's inference performance in terms of accuracy, measured through NuScenes' Detection Task Evaluation Metrics, such as mAP, mATE, mASE, mAOE, mAVE, mAAE, and NDS, as discussed in CHAPTER 2, providing insights into the effectiveness and trade-offs associated with the Posit quantization approach for this sensor fusion method.

4.1 TOOLS

4.1.1 PyTorch

PyTorch (Paszke et al., 2019) is an open-source ML library known for its flexibility, ease of use, and strong support for GPU-accelerated tensor computation. It stands out for its dynamic computational graphs, enabling more intuitive development and debugging of DL models. Deeply integrated with Python, PyTorch is easy to use and aligns well with the Python ecosystem. Its modularity and broad community support have fostered a rich ecosystem of tools and libraries, enhancing its functionality. Moreover, PyTorch offers access to a wide range of pre-trained models, facilitating rapid development and transfer learning. This combination of features makes PyTorch an effective tool for various ML applications, from academic research to industry deployment.

4.1.1.1 PyTorch's Quantization API

The PyTorch quantization API allows the optimization of NNs for efficient deployment in resource-constrained environments by reducing model size and increasing computational speed. It achieves this through different quantization strategies: Dynamic Quantization (mainly weights), Static Quantization (weights and activations with prior calibration) or Post-Training Quantization (PTQ), and Quantization-Aware Training (QAT), which integrates quantization into the training process to improve accuracy. While quantization effectively reduces model memory footprint and accelerates inference, it may sometimes lead to a slight decrease in model accuracy. The API is designed to be user-friendly, facilitating the adaptation of existing models for deployment in scenarios where low latency and reduced model size are crucial.

Developers working with PyTorch can adjust and customize the quantization process to meet their specific needs. This customization encompasses several key aspects. Firstly, it is possible to define custom quantization functions, allowing developers to apply their quantization logic selectively to different parts of a model. Additionally, PyTorch supports per-channel quantization for various types of layers such as *conv1d()*, *conv2d()*, *conv3d()*, and *linear()*.

The workflow involved in quantization includes modifying the model's module hierarchy by adding or replacing submodules. These submodules can include observers or module conversions, but crucially, the model retains its *nn.Module* structure, ensuring compatibility with broader PyTorch APIs. Developers can leverage the Quantization Custom Module API to specify the Python types corresponding to the 32-bit floating-point module, the observed module, and the quantized module, along with a configuration describing these types to be passed to the quantization APIs.

Within this framework, modules are converted according to the specified types during the preparation and conversion phases, facilitated by functions like *from_float* and *from_observed*. It is important to note that, currently, an *ObservedCustomModule* must have a single tensor output, and the framework automatically adds an observer as needed.

These customizations provide developers with greater control and flexibility over the quantization process, allowing model optimization to meet precise needs and achieve performance goals.

4.1.2 PYNQ

PYNQ (AMD, 2016) is an open-source project by Advanced Micro Devices (AMD) designed to simplify the use of ZYNQ systems, which combine an ARM processor with FPGA logic. By enabling programming with Python, PYNQ reduces the entry barrier compared to traditional hardware languages like VHDL or Verilog. Its user-friendly nature makes it ideal for education and research, allowing for rapid prototyping and experimentation. With a rich library for tasks such as data acquisition and signal processing, PYNQ supports various interfaces, making it versatile for embedded systems, ML, signal processing, and IoT applications.

Fundamental to PYNQ are its pre-built overlays, FPGA configurations for specific functions like image processing or ML accelerators, controllable directly from Python. A PYNQ overlay is a pre-designed FPGA configuration that simplifies the use of FPGA capabilities, making them accessible to developers without extensive hardware design knowledge. With seamless integration to Python, PYNQ overlays facilitate tasks like data movement, hardware configuration, and executing hardware-accelerated functions.

The ease of loading and using these overlays through simple Python commands allows users to focus on application development rather than FPGA details. PYNQ overlays are modular and reusable, enabling the combination and swapping of different overlays to create complex systems. This modularity also means they can be reused in various projects, offering flexible solutions for different applications.

Developing a PYNQ overlay involves several key steps. Initially, the specific hardware functionality is defined, such as image processing or ML acceleration. The FPGA logic is then designed using HDLs like VHDL or Verilog or HLS tools. This logic is integrated into a block design within the Vivado Design Suite, connecting the custom logic with necessary interface components like AXI interfaces.

After the block design is completed, the FPGA bitstream is generated through synthesis and implementation in Vivado. The next step involves creating the PYNQ overlay package, which includes the bitstream, hardware handoff files, and Python drivers. The Python API abstracts hardware details, providing direct methods to interact with the FPGA. Finally, the overlay is loaded onto a PYNQ-compatible board, such as the PYNQ-Z1, PYNQ-Z2, or KV260 for execution.

4.1.3 SoftPosit

The SoftPosit library (Leong, 2018) represents an open-source implementation for emulating Posit numbers. This library was developed in the C language and integrated into Python through associated extension mechanisms. Through SoftPosit, it is possible to emulate a wide variety of operations, ranging from arithmetic and logical calculations to conversions involving Posit numbers.

Natively, SoftPosit offers support for the emulation of Posits of different sizes, including 8, 16, and 32 bits, each with 0, 1, and 2 bits to represent the exponent, respectively. Additionally, the library also allows the emulation of operations in Posits with customized bit widths, ranging from 1 to 31 bits, all with 2 bits allocated for the exponent. It is important to highlight that, regardless of the number of bits selected for the representation of customized Posits, these values will be stored internally using 32 bits.

However, it is worth noting that this storage rule differs for Posits with fixed configurations of (8,0), (16,1), and (32,2). These, specifically, are stored with the precise amount of bits that their designations imply, that is, 8 bits, 16 bits, and 32 bits, respectively. It is important to note that this distinction has implications, albeit modest, on the speed of operations performed by the SoftPosit library.

Specifically, during the encoding and decoding of Posit numbers, as well as during the execution of the multiply-accumulate operation with quire, it was observed that operations performed on customized Posits, stored with 32 bits, can be up to 5% slower compared to operations executed on fixed Posits (8,0), (16,1), and (32,2).

4.2 POSITS4TORCH

Utilizing PyTorch's Quantization Custom Module API, a Posit quantization framework for DL models has been conceived, namely Posits4Torch. Posits4Torch is based on Deep PeNSieve and SoftPosit, and integrates with PyTorch through its Quantization Custom Module API.

Posits4Torch is flexible, allowing users to choose to use Posit numbers only for memory storage of module weights and biases, maintaining floating-point operations during inference (PaS) - similar to the strategy presented in Langroudi et al. (2018), Lu et al. (2019), and Langroudi et al. (2020) - or to use Posit numbers also for arithmetic calculations (PaA). When the choice is made in favor of Posit arithmetic, the original floating-point modules of the model are replaced by custom modules that support Posit numbers, resembling the strategy used by Nakahara et al. (2022). In this case, hardware acceleration in FPGA can also be utilized.

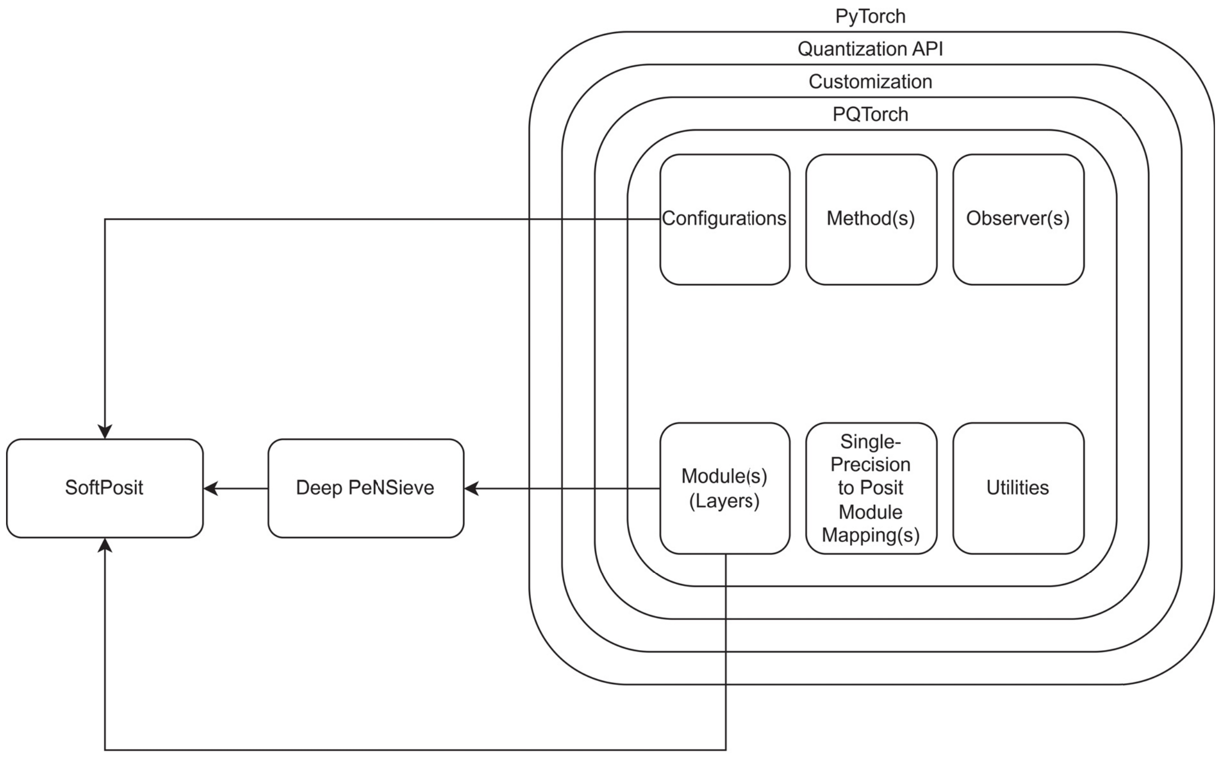
If the option is to use Posit numbers only as storage, a function is responsible for quantizing the weights and biases of the models to the desired Posit format, with all operations being executed on standard floating-point.

The structure of Posits4Torch can be visualized in FIGURE 21 and is organized as a Python package, containing internal Python modules, which can be easily installed via Package Installer for Python (PIP).

Posits4Torch is divided into six distinct modules:

1. Configuration Module ("Configurations.py"): This module includes various possible configurations for quantizing PyTorch models in Posit.

FIGURE 21 – STRUCTURE OF THE POSIT QUANTIZATION API.



SOURCE: The Author.

2. Layer Module ("Quantized.py"): Here are the custom PyTorch modules that support Posit numbers during inference, based on Deep PeNSieve for CPU-emulated Posit arithmetic, or on Posits4TorcHA for FPGA-based Posit arithmetic.

3. Methods Module ("Quantization.py"): This module contains the methods responsible for executing the quantization process.

4. Utilities Module ("Utilities.py"): This module contains declarations of global variables, constants, and various auxiliary methods and macros.

5. Observers Module ("Observers.py"): This module includes class declarations responsible for determining the quantization scale and zero-point, allowing the selection of the best quantization formats for a given module based on a sample input. It is important to note that active observers are optional and not currently used; however, the PyTorch quantization API requires some type of observer. Therefore, a "Lazy"or passive observer was created, which does not interfere with the quantization process.

6. Remote Hardware Acceleration Module ("RemoteAccel.py"): This module uses the Hypertext Transfer Protocol (HTTP) protocol for communication - through an HTTP client - with an HTTP server on the remote device where the Posit hardware accelerator responsible for executing operations during inference via PYNQ is deployed.

Additionally, modifications were made to both Deep PeNSieve and SoftPosit.

The modifications to Deep PeNSieve were limited to changing the structure of the original Python project so that Deep PeNSieve can be installed via PIP and, therefore, used globally by other Python programs and packages, such as Posits4Torch.

The modifications to SoftPosit include the creation of explicit types for Posit numbers ([1, 31], 2), making their use simpler and more user-friendly. Previously, the library supported these formats only through a single type, *Posit_2*, which needed to be instantiated with the number of representation bits at runtime, making its use impractical in conjunction with Posits4Torch. Therefore, explicit types such as *posit8_2*, equivalent to a Posit (8, 2), were created to facilitate the manipulation of these numbers.

4.2.1 Posits4TorcHA: Inference Acceleration on FPGA

With the aim of accelerating inference using Posit through the use of an FPGA, an AMD Kria KV260 Vision AI Starter Kit development board was used to implement a hardware accelerator consisting of an AXI Direct Memory Access (DMA)-compliant 2D Posit MAC Unit Array PYNQ Overlay. The open-source PYNQ framework was employed to facilitate interaction with the FPGA.

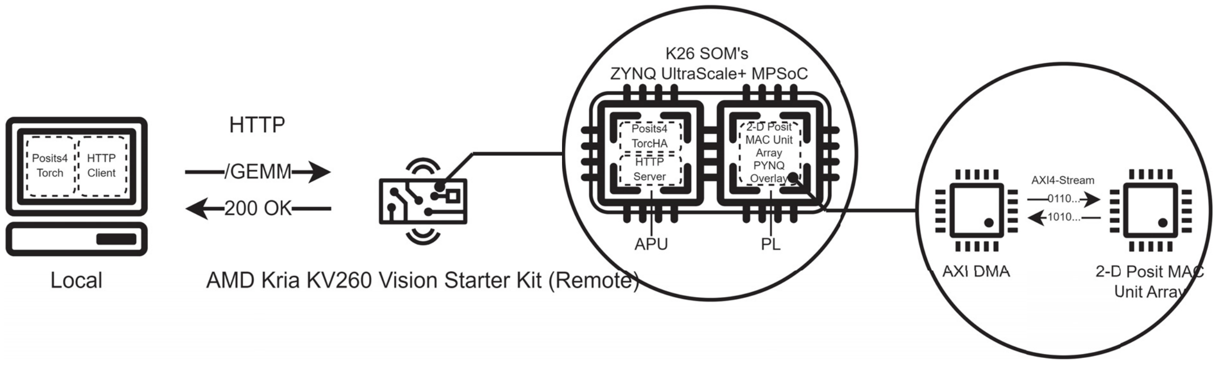
The main focus of the acceleration is on General Matrix Multiplication (GEMM) operations. This effort aims to enhance the efficiency of Posit NN computations, contributing to improved performance of DL models used in Computer Vision (CV) applications, such as CenterFusion.

As mentioned earlier, the hardware acceleration provided by Posits4TorcHA is based on an AXI DMA-compliant 2D Posit MAC Unit Array PYNQ Overlay. Developed in RTL using VHDL, this array is highly parameterized. It can be customized based on various parameters, such as the number of rows and columns in the array, the precision of Posit numbers (in bits), the bit-width of the exponent field in the Posit representation, the size of the Posit MAC Units' accumulator, the depth of the output First-In First-Out (FIFO) buffers, and the model of the Posit MAC Units. This flexibility allows adapting the array to specific computational needs, optimizing performance for different matrix sizes and precisions. To maximize performance, Posits4TorcHA incorporates a Cython module that accelerates the processing and data communication between the software on the Application Processing Unit (APU) of the edge device and its respective array of MAC Units in the Programmable Logic (PL), through compiled C code and parallel processing. This module optimizes the speed of read/write operations to and from the PYNQ buffers in the device's Random Access Memory (RAM), which is essential for efficient data transfer when using AMD's AXI DMA Intellectual Property (IP) in PYNQ overlays, as it is the case here. By accelerating these operations, the Cython module ensures that hardware acceleration is as effective as possible.

The package also includes an HTTP server, which processes requests from the Posits4Torch HTTP client package. This server-client configuration is crucial for executing GEMM operations remotely, allowing heavy computational work to be handled by dedicated hardware rather than the local CPU.

A complete overview of Posit4Torch / Posits4TorcHA functionallity in conjunction with an AMD Kria KV260 Vision Starter Kit device, as described earlier, can be seen in FIGURE 22.

FIGURE 22 – FUNCTIONALLITY OF POSITS4TORCH / POSITS4TORCHA WITH AN AMD KRIA KV260 VISION STARTER KIT.



SOURCE: The Author.

Additionally, Posits4TorcHA offers off-the-shelf PYNQ overlay implementations specifically synthesized for the AMD Kria KV260 Vision Starter Kit, which by itself is composed of a Kria K26 System-On-Module (SOM) that has an embedded ZYNQ UltraScale+ MPSoC. These implementations use a 2D Posit MAC Unit Array that is compliant with the AXI4-Stream protocol and whose underlying Posit MAC Units design is based on the work of Crespo et al. (2023), ensuring efficient data streaming and processing. The overlays provided for this FPGA platform enhances its capability to handle complex matrix multiplications efficiently, making it a possible solution for DL and other data-intensive applications. Nonetheless, PYNQ overlays for hardware platforms other than the AMD Kria KV260 Vision Starter Kit can be synthesized using AMD's Vivado Design Suite along with the project files, which include VHDL and/or Verilog descriptions for both the AXI4-Stream-compliant Posit MAC Unit wrapper and the 2D Posit MAC Unit Array in conjunction with their respectives parameterized IP packages, and the top-level block diagram for the overlay.

4.2.1.1 Development of the Posit MAC Unit

The Posit MAC Unit is a specialized hardware component developed to perform arithmetic operations using Posits. Developed in RTL using VHDL within the Vivado Design Suite 2022.2, this Posit MAC Unit acts as an AXI4-Stream encapsulation for previously developed Posit MAC Unit models documented in existing literature, such as in (Murillo et al., 2021), (Nakahara et al., 2022), and (Crespo et al., 2023), which were designed to efficiently handle MAC operations in Posit numbers.

Key parameters of the Posit MAC Unit include its precision and bit width (N), which defines how many bits are used to represent each Posit number. The exponent bit width (Es) specifies the length of the exponent field in the Posit format, helping to adjust the dynamic range. The size of the accumulator (or quire) ($QSize$) is another crucial parameter, representing the bit width of this accumulator, allowing for the exact accumulation of intermediate results without rounding. The output FIFO buffer depth ($Depth$) indicates the storage capacity for the output results of the MAC operations, ensuring efficient data handling. The model ($Model$) specifies the underlying architecture for the Posit MAC Unit.

The unit communicates through two main channels: the AXI4-Stream slave interface and the AXI4-Stream master interface. The slave interface, with a data width of $2 \cdot \max(8, 2^{\lceil \log_2(N) \rceil})$ bits, carries the input data to the MAC Posit unit. These consist of a sequence of pairs of Posit numbers (N, Es), which is further subdivided into a number of segments less than or equal to the depth of the MAC Unit's FIFO buffer. On the other hand, the master interface, with a data width of $\max(8, 2^{\lceil \log_2(N) \rceil})$ bits, carries the output data from the Posit MAC Unit. These are organized into a sequence of Posit numbers (N, Es) accumulated in the quire and stored in the output FIFO buffer, each number corresponding to a given input segment.

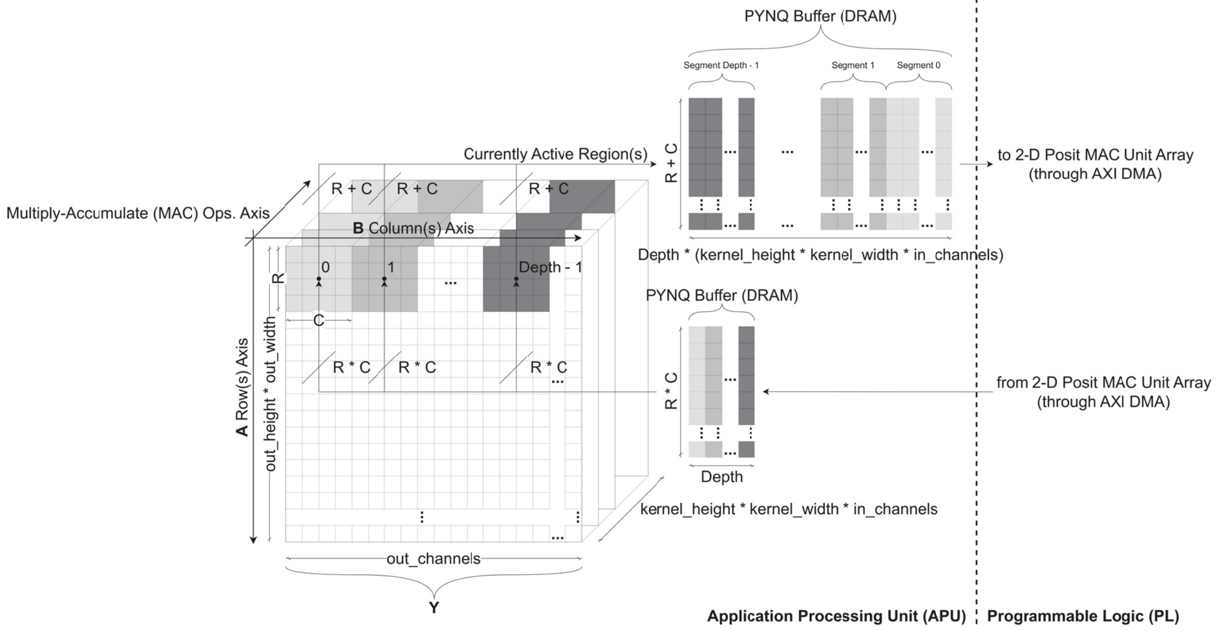
The central functionality of the MAC Unit involves multiplying pairs of Posit numbers and accurately accumulating the results using the quire. This ensures high precision in the accumulation process, reducing errors typically associated with floating-point arithmetic. The unit's parameterized design allows extensive customization, making it adaptable to various precision requirements and hardware configurations. Its compliance with AXI4-Stream interfaces ensures compatibility with standard data communication protocols, facilitating integration into larger systems.

4.2.1.2 Development of the 2D Posit MAC Unit Array

The 2D Posit MAC Unit Array is a digital design developed to enhance the efficiency of matrix operations, specifically matrix multiplication, using the Posit format. Developed using Vivado Design Suite 2022.2 at the RTL level with VHDL, this array of units is characterized by its parameterized design, allowing flexibility and customization to meet various computational requirements. The key parameters include the number of rows (R) and columns (C) in the array, the precision or bit-width of the Posit format (N), the bit-width of the exponent field (Es), the size of the MAC Unit accumulators ($QSize$), the depth of the output FIFO buffer ($Depth$), and the specific model of these Posit MAC Units ($Model$).

This array complies with the AXI4-Stream protocol, ensuring high-speed data transfer capabilities. It includes both slave and master interfaces. The slave interface carries input data to the array of MAC Units, consisting of a sequence of tuples of $R + C$ Posit $(N,\ Es)$ numbers. This sequence of tuples is subdivided into a number of segments corresponding to the output FIFO buffer depth configuration of the Posit MAC Units in the array. These input segments, in turn, have a size equal to the number of columns in the input matrix A (or the number of rows in the input matrix B), where matrices A and B are the operands of a given matrix multiplication operation to be performed. Each of these segments corresponds to a currently active region in the output matrix, Y, whose elements are computed through a dot product operation between a specific row of matrix A and a specific column of matrix B. The master interface, similarly configured, carries the output data from the array of MAC Units, which is a sequence of tuples of $R \cdot C$ Posit $(N,\ Es)$ numbers representing the results of the multiplication and accumulation (dot-product) operations of each of the Posit MAC Units in the 2D Posit MAC Unit Array, for each input segment. This means that the sequence of output tuples is sized to match the depth of the output FIFO buffers. A visual representation of this can be seen in FIGURE 23.

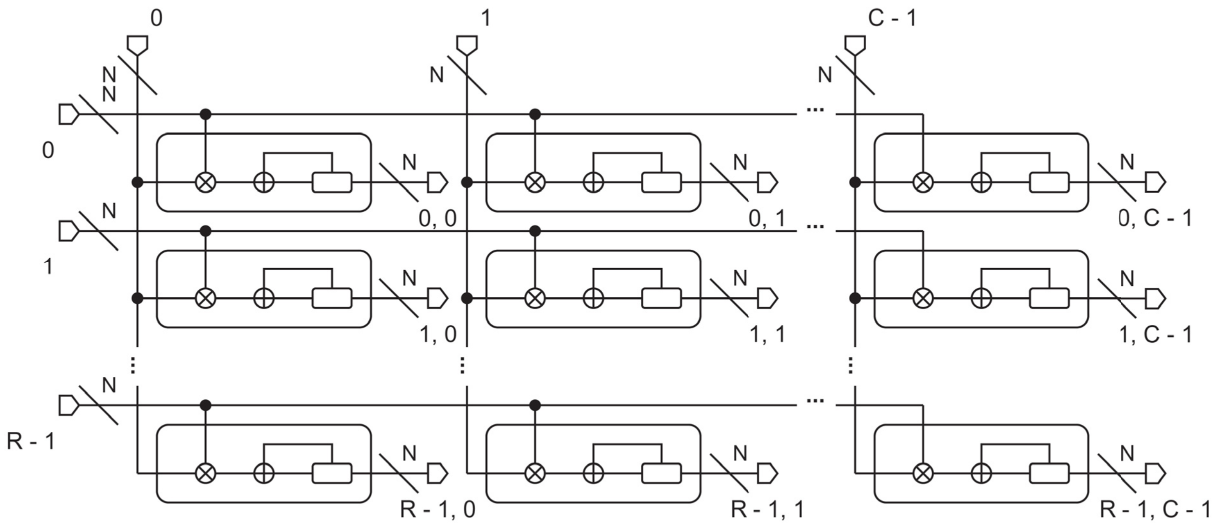
FIGURE 23 – REPRESENTATION OF A 2D CONVOLUTION OPERATION USING GENERAL MATRIX MULTIPLICATION (GEMM) THROUGH POSITS4TORCHA'S AXI DMA-COMPLIANT 2D POSIT MAC UNIT ARRAY PYNQ OVERLAY.


PYNQ Buffer (DRAM)
Segment Depth - 1
Segment 1
Segment 0
R + C
to 2-D Posit MAC Unit Array (through AXI DMA)
Currently Active Region(s)
Multiply-Accumulate (MAC) Ops. Axis
R + C
R + C
R + C
B Column(s) Axis
0
1
Depth - 1
R
C
Depth * (kernel_height * kernel_width * in_channels)
PYNQ Buffer (DRAM)
R * C
R * C
R * C
R * C
from 2-D Posit MAC Unit Array (through AXI DMA)
A Row(s) Axis
out_height * out_width
Depth
kernel_height * kernel_width * in_channels
out_channels
Y
Application Processing Unit (APU)
Programmable Logic (PL)

SOURCE: The Author.

Functionally, each Posit MAC Unit within the array performs the multiplication and accumulation of Posit numbers. The 2D structure of the array, as seen in FIGURE 24, allows for the simultaneous processing of $R \cdot C$ dot products, each in a MAC Unit, significantly enhancing computational efficiency. The results of these operations for each input segment are stored in the output FIFO buffer of each Posit MAC Unit, ensuring availability for subsequent processing.

FIGURE 24 – STRUCTURE OF THE 2D POSIT MAC UNIT ARRAY.



SOURCE: The Author.

The 2D Posit MAC Unit Array is implemented on FPGA, leveraging the reconfigurable nature of this device to optimize performance for specific applications. This implementation supports hardware acceleration for GEMM tasks, making the array particularly suitable for high-performance computing applications such as ML and DL inference. By using the Posit format, the design aims to provide superior numerical precision and dynamic range compared to traditional floating-point systems, which is crucial in computational tasks.

The developed array is then included in a PYNQ overlay - along with an AMD AXI DMA IP and auxiliary AMD AXI infrastructure IPs - to be loaded and executed on an FPGA through the Posits4TorcHA package. The development of the overlay also took place in Vivado Design Suite 2022.2.

4.3 CENTERFUSION QUANTIZATION

The case study on CenterFusion explores the integration of Posits4Torch to enhance DL performance through Posit quantization and remote hardware acceleration. CenterFusion is primarily developed in Python using PyTorch. By incorporating Posits4Torch, Posit quantization of CenterFusion's parameters can be achieved.

CenterFusion's source code was modified to leverage Posits4Torch for both quantization and inference of its primary and secondary regression heads. This modification involves two main methods: Posit-as-Storage (PaS) and Posit-as-Arithmetic (PaA). In the PaS method, CenterFusion's regression heads parameters are quantized using the Posit format. During inference, these Posit values are converted back to floating-point format, allowing the use of floating-point arithmetic on a NVIDIA GeForce RTX 3090 GPU. Various Posit configurations, such as Posit (8, 0), Posit (16, 1), Posit (32, 2), and Posit ([2, 31], 2), are supported.

The PaA method enables remote hardware acceleration for Posit arithmetic operations. This setup uses a PYNQ overlay running on an AMD Kria KV260 Vision Starter Kit, with a 2D 8x8 Posit MAC Units Array with an output FIFO buffer depth of 512. The underlying model of the Posit MAC Units is based on the work of (Crespo et al., 2023). The overlay operates on the PL of a ZYNQ UltraScale+ MPSoC from a K26 SOM, at a clock speed of 100 MHz, significantly enhancing computational efficiency for Posit arithmetic operations in CenterFusion compared to its single-threaded-CPU-emulated-Posit-quantized counterpart via Deep PeNSieve on an AMD EPYC 7413 CPU.

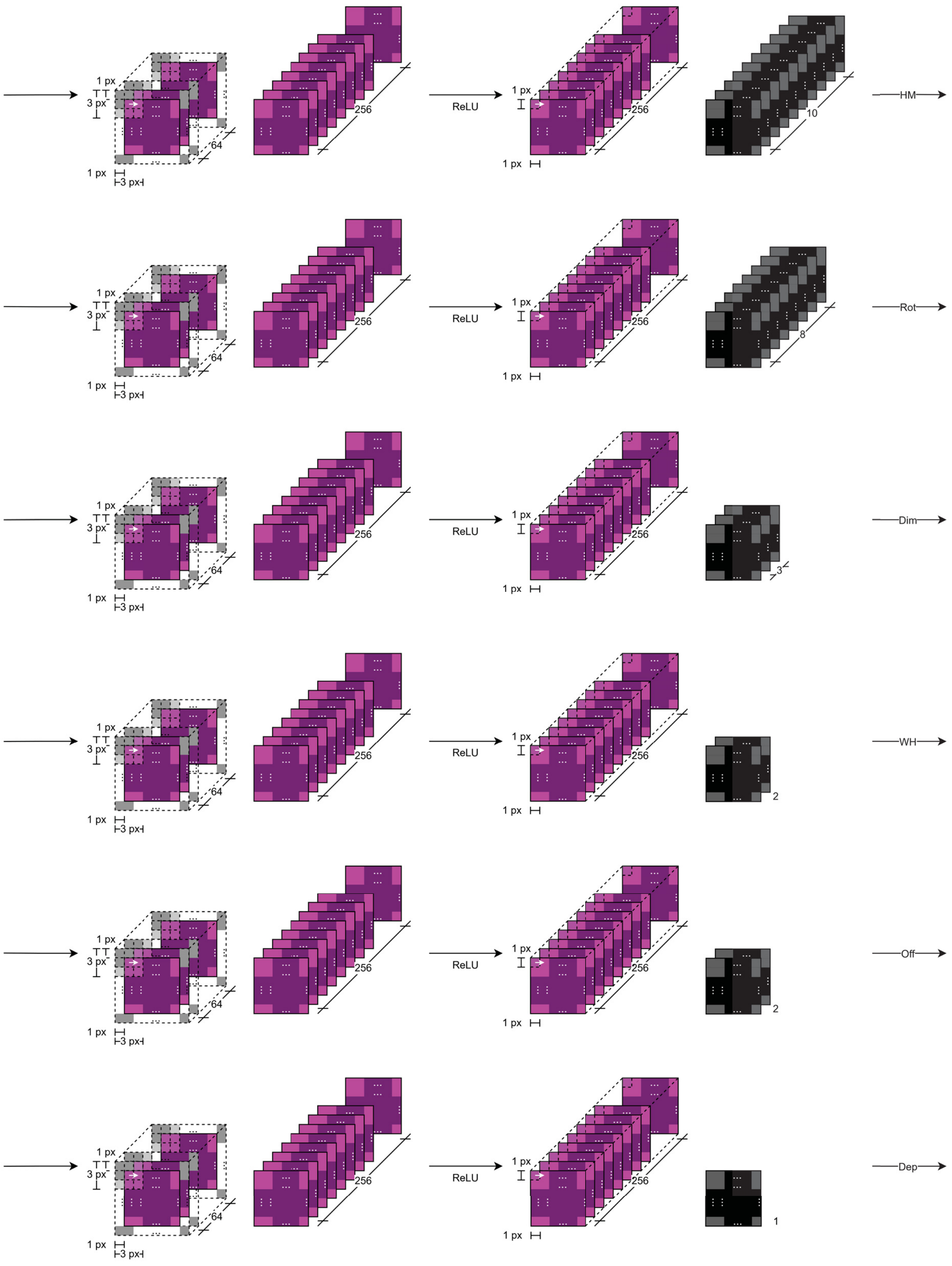
However, Posits4Torch does not yet support DCNs (Dai et al., 2017), which are essential for CenterFusion's backbone. Consequently, Posit quantization of CenterFusion's backbone is not yet achievable. Despite this limitation, this integration allows for Posit quantization and inference of CenterFusion's regression heads, providing insights into the network's overall performance impact.

To execute and test the Posit-quantized CenterFusion model, the mini-validation subset of the NuScenes dataset is used. The results are evaluated using the NuScenes Detection Task Evaluation Metrics, as discussed in CHAPTER 2. These metrics help assess the impact on inference performance brought by Posit quantization and hardware acceleration.

CenterFusion's regression heads consist of convolutional layers with ReLU activation. Each primary regression head consists of two convolutional layers. The first layer has 64 input channels, 256 output channels, and a 3x3 pixel kernel, with a stride of 1 pixel and padding of 1 pixel. The second convolutional layer has 256 input channels, a 1-pixel kernel, and a stride of 1 pixel (no padding). The number of output maps varies according to the property to be regressed, such as heatmaps, rotations, dimensions, heights, widths, offsets, and object depths. These primary regression heads can be visualized in FIGURE 25.

The secondary regression heads differ from the primary regression heads and consist of four consecutive convolutional layers. The first layer is similar to that of the primary regression heads. The second and third layers have 256 input and output channels, a 1-pixel kernel size, and a stride of 1 pixel, with no padding. The final layer relates to the property being regressed, and also has 256 input channels, a 1-pixel kernel size, and a stride of 1 pixel (no padding). The output channels of this final layer correspond to the maps associated with specific object properties to be detected. The

FIGURE 25 – PRIMARY REGRESSION HEADS OF CENTERFUSION.



SOURCE: The Author.

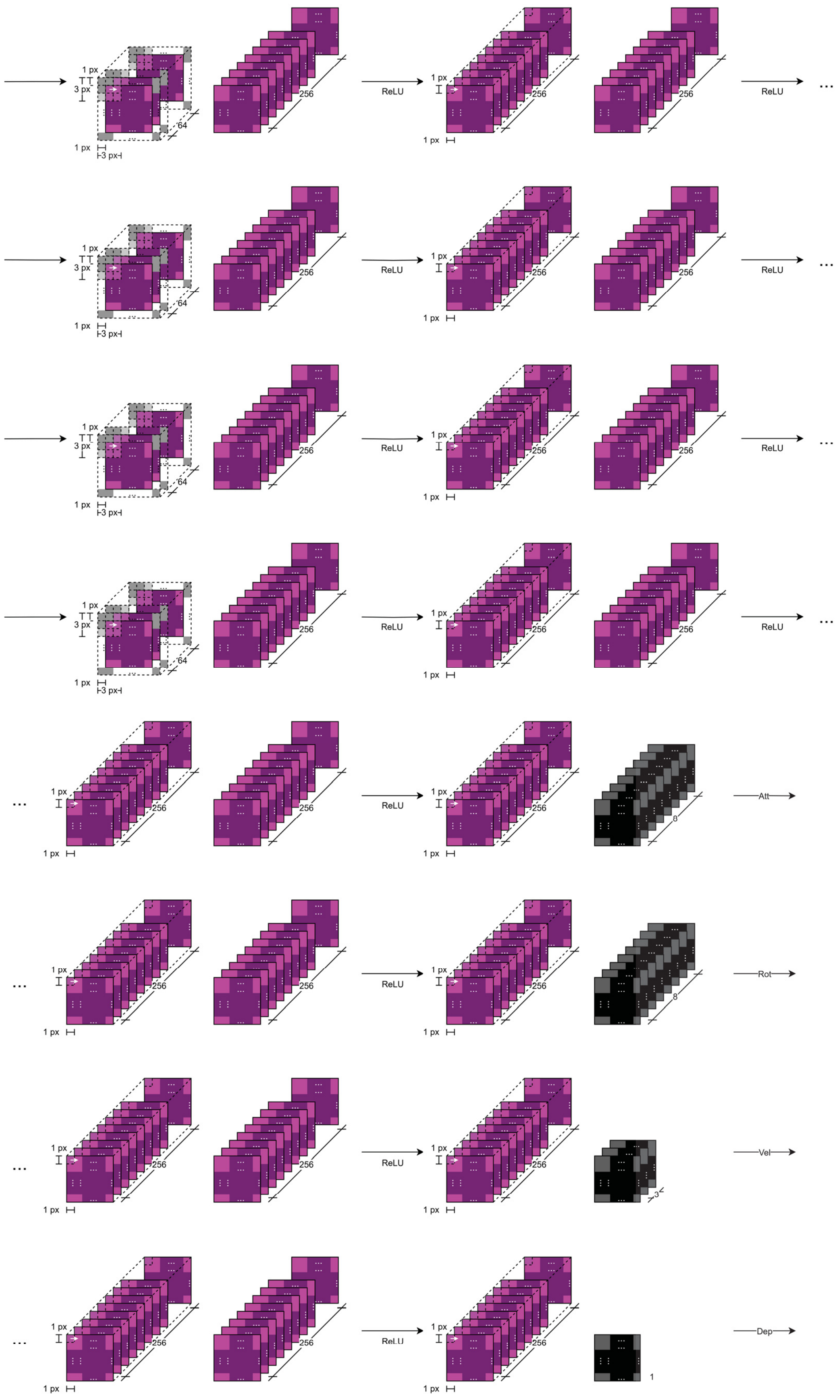
number of output maps varies depending on the property to be computed, such as object class attributes, velocity, re-estimated rotation, and re-estimated depth. CenterFusion's secondary regression heads can be seen in FIGURE 26.

CenterFusion's quantization process effectively occurs during the initial stages before the execution of the model's inference. In order to do this, CenterFusion's source code was modified. These modifications added Posit quantization capabilities to CenterFusion's regression heads through customization options. New command-line arguments were added to CenterFusion for this purpose. Specifically, the additional arguments are:

- *–quantize_heads*, indicating that the user wishes to perform the regression heads quantization process;
- *–N <N>*, the bit-width of the associated Posit format, in the range of [1, 32];
- *–Es <es>*, the maximum number of bits that the exponent can have in the Posit number, belonging to the set {0, 1, 2};
- *–qdevice*, the device on which to run the Posit-quantized CenterFusion inference (applies only to the PaA strategy);
- *–fpga_host*, the Internet Protocol Version 4 (IPv4) address of the FPGA device on which the inference will be executed (applies only to the PaA strategy on FPGA);
- *–fpga_port*, the Transport Control Protocol (TCP)/User Datagram Protocol (UDP) port of the device on which inference will be executed on FPGA, corresponding to the associated HTTP server (applies only to the PaA strategy on FPGA);
- *–fpga_conf*, Posits4TorcHA's AXI DMA-compliant 2D Posit MAC Unit Array PYNQ Overlay configurations (applies only to the PaA strategy on FPGA);
- *–inference_num_workers <num_workers>*, the number of APU cores on the inference device to be used by Cython to do R/W operations to and from PYNQ Buffers on DRAM (applies only to the PaA strategy on FPGA).

After initialization of the inference process, quantization will occur following CenterFusion's model instantiation step, obeying the command-line arguments as described previously. A visual representation of CenterFusion's regression heads quantization process can be seen in FIGURE 27.

FIGURE 26 – SECONDARY REGRESSION HEADS OF CENTERFUSION.



SOURCE: The Author.

FIGURE 27 – POSIT QUANTIZATION OF CENTERFUSION.

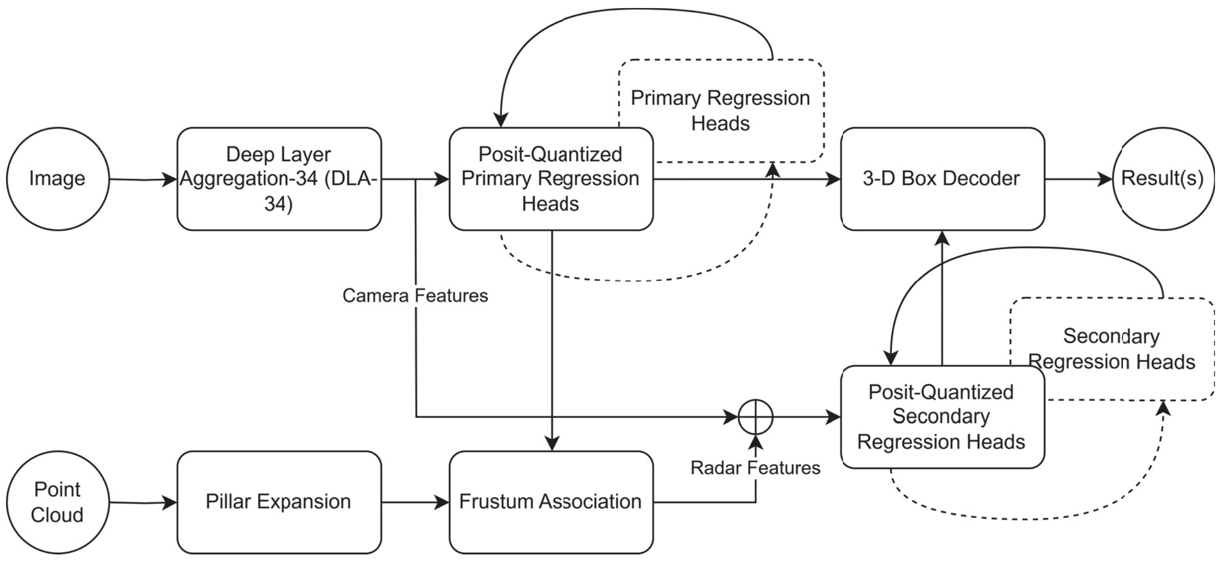


Image
Deep Layer Aggregation-34 (DLA-34)
Posit-Quantized Primary Regression Heads
Primary Regression Heads
3-D Box Decoder
Result(s)
Camera Features
Secondary Regression Heads
Posit-Quantized Secondary Regression Heads
Radar Features
Point Cloud
Pillar Expansion
Frustum Association

SOURCE: The Author.

5 RESULTS

5.1 VALIDATION OF POSITS4TORCH'S / POSITS4TORCHA'S FUNCTIONALITY

Initially, the functionality of Posits4Torch and Posits4TorcHA was validated through extensive testing. Posits4Torch allows for the quantization of PyTorch models using PyTorch's Quantization Custom Module API, while Posits4TorcHA provides an AXI DMA-compliant 2D Posit MAC Unit Array PYNQ Overlay for the AMD Kria KV260 Vision Starter Kit.

The tests focused on Posit (8, 2) quantization for both the Deep PeNSieve's single-threaded-CPU-emulated backend, running on an AMD EPYC 7413 CPU, and Posits4TorcHA's FPGA-based backend, utilizing an AMD Kria KV260 Vision Starter Kit running an AXI DMA-compliant 2D 8x8 Posit MAC Unit Array PYNQ Overlay. The tested model, similar to CenterFusion's secondary regression heads, was fed with a 4-D 2x64x80x45 random input tensor. The results in TABLE 9 indicate, as expected, that the quantization-induced degradation in the inference accuracy for both the single-threaded-CPU-emulated Posit model from Deep PeNSieve and the FPGA-based Posit model from Posits4TorcHA are identical since both use exact Posit arithmetic. Notably, the inference time for the FPGA-based model is almost 1000 times lower than that of the CPU-emulated model, but approximately 300 times higher than PyTorch's GPU-based 32-bit floating-point model running on a NVIDIA GeForce RTX 3090. These findings confirm the superiority of the FPGA-based model over the CPU-emulated model in terms of performance and efficiency, although it still has lower performance than the original GPU-based model.

TABLE 9 – POSITS4TORCH / POSITS4TORCHA FUNCTIONALITY VALIDATION RESULTS.

Backend	**Inference Error (%)**	**Inference Time (s)**
CPU (AMD EPYC 7413)	4.97	3194.69
FPGA (AMD Kria KV260 Vision Starter Kit)	4.97	3.49

SOURCE: The Author.

AMD's Kria KV260 Vision Starter Kit PL resource consumption for the 2D 8x8 Posit MAC Unit Array was evaluated using Vivado Design Suite Synthesis Utilization Report and Place Design Utilization Report. The results - given in terms of LUTs , Block RAM (BRAM), Ultra RAM (URAM), and DSP Slices - are detailed in TABLE 10 for the resource utilization exclusively of the 2D 8x8 Posit MAC Unit Array after synthesis and in TABLE 11 for the complete PYNQ overlay, which includes not only the 2D 8x8 Posit MAC Unit Array but also AMD's AXI DMA IP and AMD's AXI Infrastructure IPs, after placement and routing. The total on-chip power consumption, as reported in the Vivado Routing Design Power Report, increases, though marginally, with higher Posit precisions, being 2.939 W, 3.033 W, and 3.090 W for 6, 7, and 8-bit precisions, respectively.

TABLE 10 – PROGRAMMABLE LOGIC (PL) RESOURCE CONSUMPTION OF THE 2D 8X8 POSIT MAC UNIT ARRAY AFTER SYNTHESIS.

Precision N	**LUT**	**BRAM**	**URAM**	**DSP**
6-bit	58283	0	0	0
7-bit	71163	0	0	0
8-bit	78129	32	0	0

SOURCE: The Author.

TABLE 11 – PROGRAMMABLE LOGIC (PL) RESOURCE CONSUMPTION OF POSITS4TORCHA'S AXI DMA-COMPLIANT 2D POSIT MAC UNIT ARRAY PYNQ OVERLAY, AFTER PLACEMENT AND ROUTING.

Precision N	**LUT**	**BRAM**	**URAM**	**DSP**
6-bit	64948	17	0	0
7-bit	77903	17	0	0
8-bit	85060	49	0	0

SOURCE: The Author.

5.2 CENTERFUSION CASE STUDY

In this section, the results obtained for the CenterFusion Case Study are discussed for both the PaS and PaA strategies, respectively.

5.2.1 Posit-as-Storage (PaS)

For the PaS strategy, the results from the evaluation of CenterFusion on the NuScenes dataset after inference using both regression heads quantized through Posits4Torch using Posit (8, 0), (16, 1), and (32, 2) are available in TABLE 12. These metrics are presented for several Posit quantization formats as well as for the original 32-bit floating-point baseline format. As expected, larger bit-widths imply a more precise approximation of the values to the baseline values for each metric, obtained using floating-point 32-bit, with the worst case scenario being that of the Posit (8, 0), whose degradation in the mAP and NDS are, respectively, of 5.6% and 3.8% from TABLE 12.

Similar results from the sole quantization of the primary regression heads are available in TABLE 13. In this case, the degradation is less perceptible. The degradations

TABLE 12 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (8, 0), (16, 1), AND (32, 2), USING POSIT-AS-STORAGE (PAS).

Format	mAP (-)	mATE (m)	mASE (-)	mAOE (rad)	mAVE (m/s)	mAAE (-)	NDS (-)
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (8, 0)	0.258	0.766	0.455	0.595	0.778	0.304	0.339
Posit (16, 1)	0.311	0.666	0.463	0.596	0.767	0.308	0.376
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

for the mAP and NDS are, respectively, of 0.7% and 0.2% for the 8-bit Posit quantization, as per TABLE 13.

TABLE 13 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (8, 0), (16, 1), AND (32, 2), FROM THE QUANTIZATION OF THE PRIMARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).

Format	mAP (-)	mATE (m)	mASE (-)	mAOE (rad)	mAVE (m/s)	mAAE (-)	NDS (-)
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (8, 0)	0.307	0.675	0.457	0.596	0.752	0.305	0.375
Posit (16, 1)	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

Finally, the results from the individual quantization of the secondary regression heads are available in TABLE 14. Here, the degradations of 5.4% in the mAP and 3.9% in the NDS for an 8-bit Posit precision resembles that which was obtained when quantizing both regression heads simultaneously.

TABLE 14 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (8, 0), (16, 1), AND (32, 2), FROM THE QUANTIZATION OF THE SECONDARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).

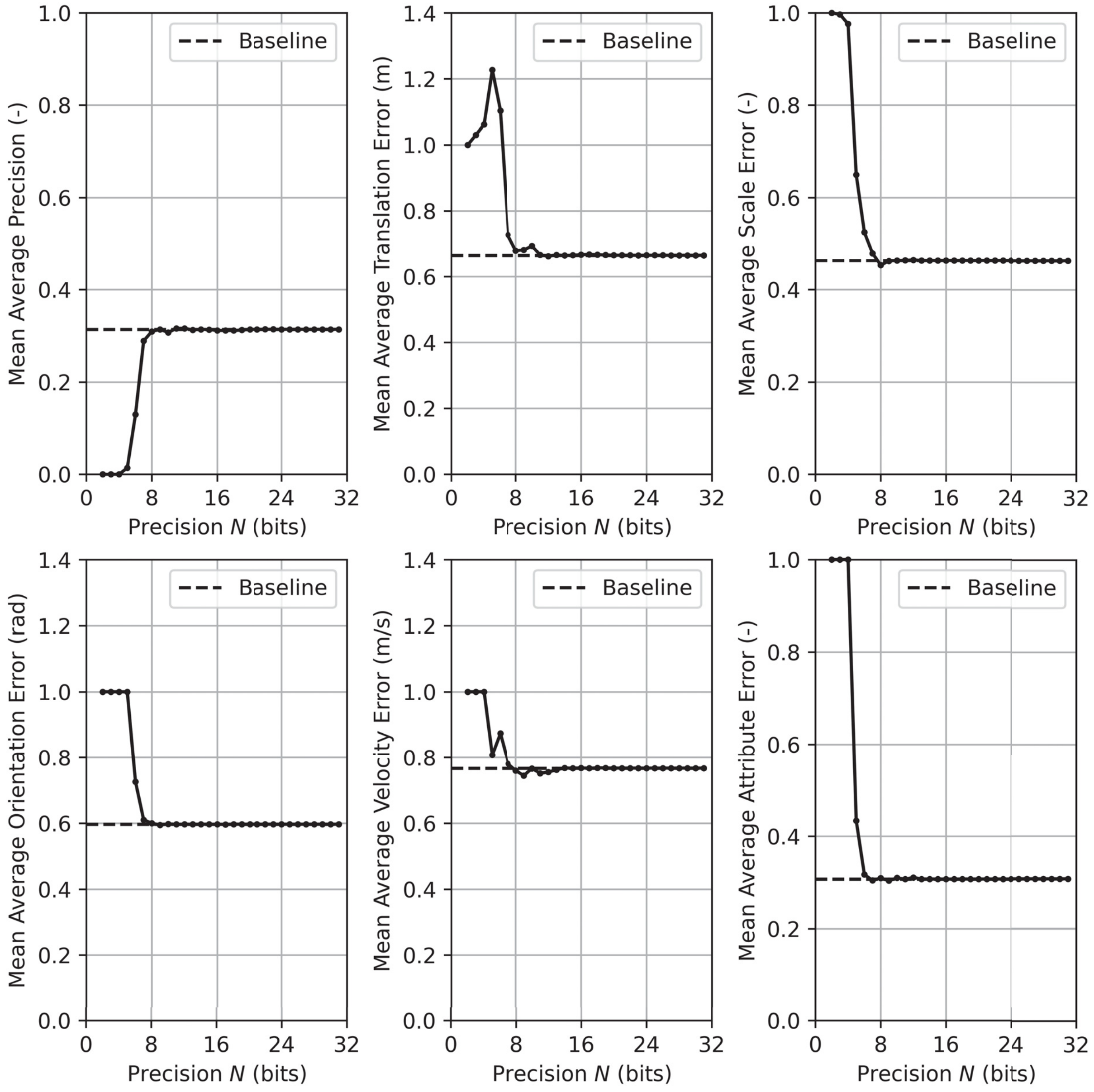
Format	mAP (-)	mATE (m)	mASE (-)	mAOE (rad)	mAVE (m/s)	mAAE (-)	NDS (-)
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (8, 0)	0.260	0.761	0.460	0.603	0.793	0.304	0.338
Posit (16, 1)	0.311	0.666	0.463	0.596	0.767	0.308	0.376
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

Furthermore, these same metrics can be visualized in the graphs shown in FIGURE 28 and FIGURE 29 when Posits (N, 2) with a variable number of bits are used for the joint quantization of CenterFusion's regression heads. The specific values for each bit width, in this case, can be read from TABLE 15. It is found that Posit (7,

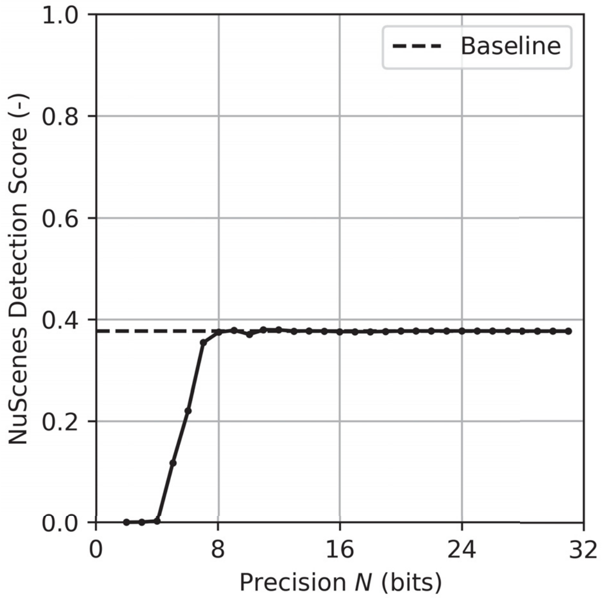
2) quantization is sufficient to achieve results comparable to those obtained with both higher-bit-width Posit quantizations and 32-bit floating point numbers, with a degradation in mAP of 2.5% and in NDS of 2.3% when compared to the baseline, as per TABLE 15.

FIGURE 28 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

FIGURE 29 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

TABLE 15 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2), FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).

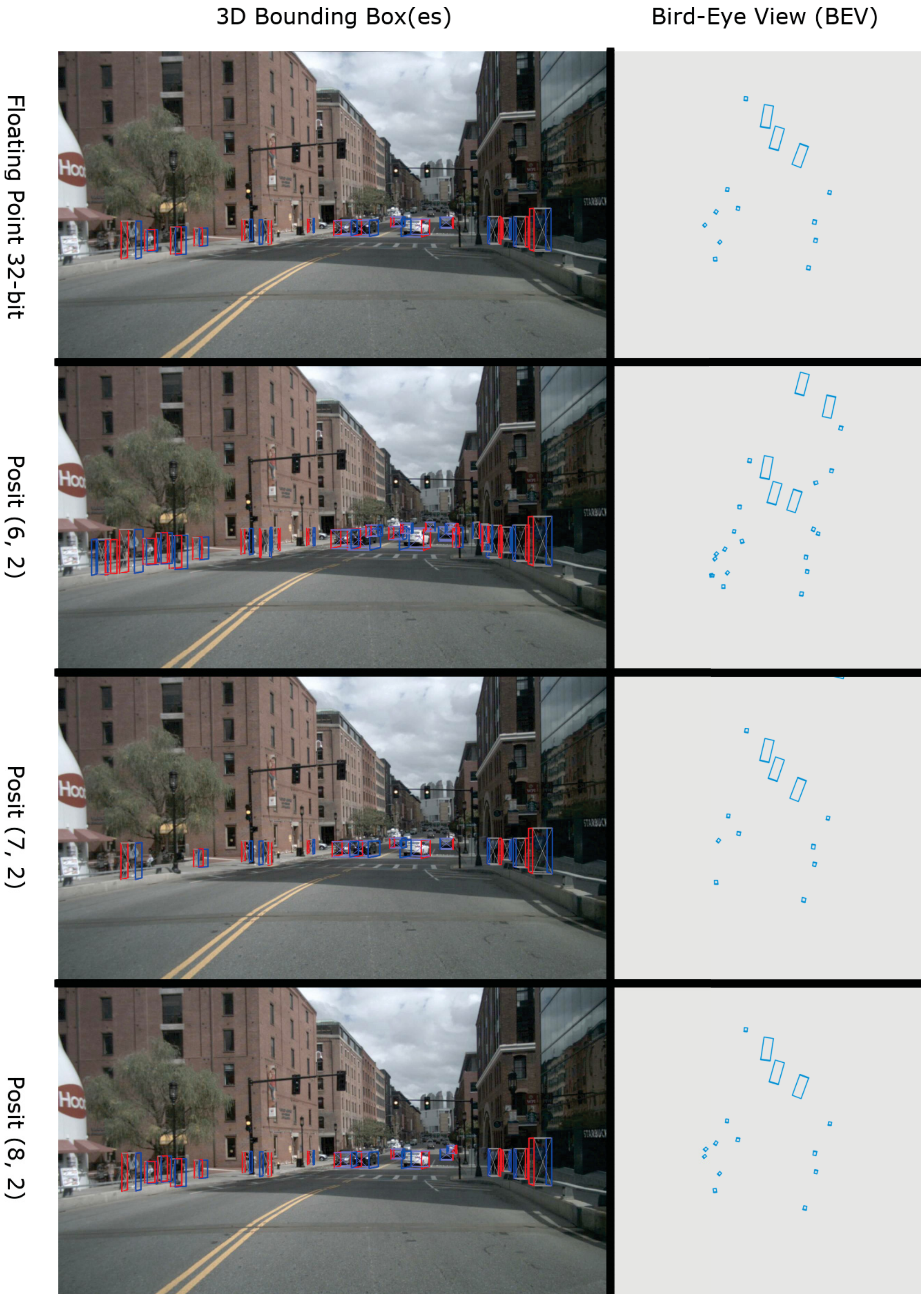
Format	**mAP (-)**	**mATE (m)**	**mASE (-)**	**mAOE (rad)**	**mAVE (m/s)**	**mAAE (-)**	**NDS (-)**
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (2, 2)	0.000	1.000	1.000	1.000	1.000	1.000	0.000
Posit (3, 2)	0.000	1.030	0.997	1.000	1.000	1.000	0.000
Posit (4, 2)	0.000	1.062	0.976	1.000	1.000	1.000	0.002
Posit (5, 2)	0.014	1.227	0.650	1.000	0.808	0.434	0.118
Posit (6, 2)	0.130	1.103	0.525	0.725	0.874	0.317	0.221
Posit (7, 2)	0.289	0.727	0.479	0.610	0.781	0.305	0.354
Posit (8, 2)	0.309	0.678	0.453	0.600	0.760	0.310	0.375
Posit (16, 2)	0.311	0.666	0.463	0.596	0.767	0.308	0.376
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

Additionally, it is important to mention that Posit (7, 2) quantization demonstrates superior performance compared to Posit (8, 0) quantization, which showed a NDS degradation of 3.9% in the worst case when compared to 32-bit floating point, despite having a larger bit-width. This highlights the influence of the exponent field in this case. A notable observation, derived from analyzing the curves for the mAVE and mATE metrics from FIGURE 28, is that especially for the mATE metric and exclusively for the PaS strategy, smaller bit-widths appear to exhibit superior performance compared to larger bit-widths, up to 7 bits, although they are still considerably inferior to quantizations with bit-widths equal to or greater than 7 bits. For those who cannot afford any reduction in precision, Posit (8, 2) quantization emerges as a viable alternative. This format resulted in an NDS degradation of only 0.2%. This approach enables a substantial reduction of up to 75% in weights storage size in memory, compared to the original model size. Consequently, this results in an equivalent reduction in the bandwidth required for memory read and write operations, since it uses 1/4 of the original number of bits per parameter. Performing inference with these quantization settings presents notable advantages from a hardware perspective, especially in terms of resources like logic units in FPGAs and area in ASICs, as Posit arithmetic operators can operate with reduced bit-widths.

Finally, as part of the qualitative results, FIGURE 30 illustrates the 3D bounding box predictions and bird's-eye views generated by the Posit-quantized CenterFusion model on a single sample frame from the NuScenes minival dataset, using Posit precision levels of 6, 7, and 8 bits. The results are shown for both the 32-bit floating-point baseline and Posit precisions at 6, 7, and 8 bits.

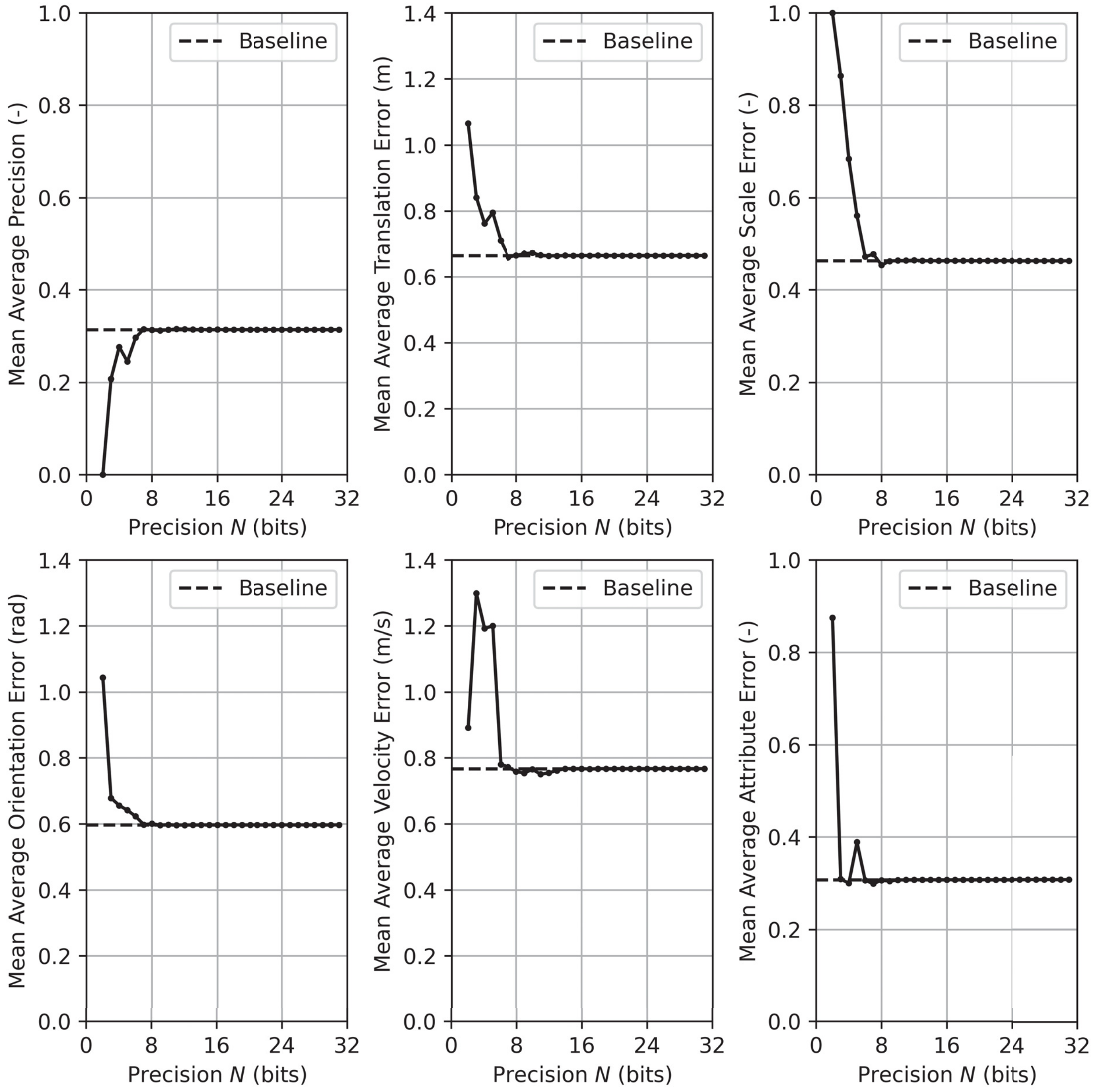
FIGURE 30 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

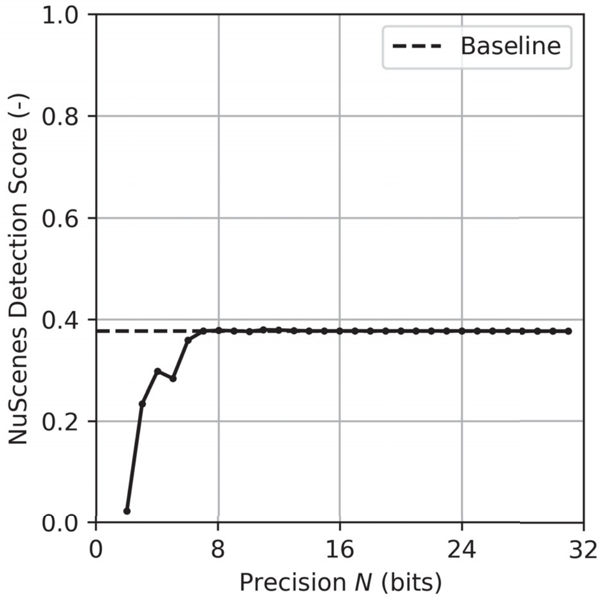
Also, in FIGURE 31 and FIGURE 32 the curves obtained for these same metrics can be visualized when solely CenterFusion's primary regression heads are quantized using Posit (N, 2). Again, the specific values for each bit width can be read from TABLE 16. In this case, the Posit (7, 2) quantization does not show any perceptible degradation to the NDS and mAP when compared to the baseline, and the Posit (6, 2) quantization degrades the mAP on 1.8% and the NDS on 1.8%, as per TABLE 16.

FIGURE 31 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

FIGURE 32 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

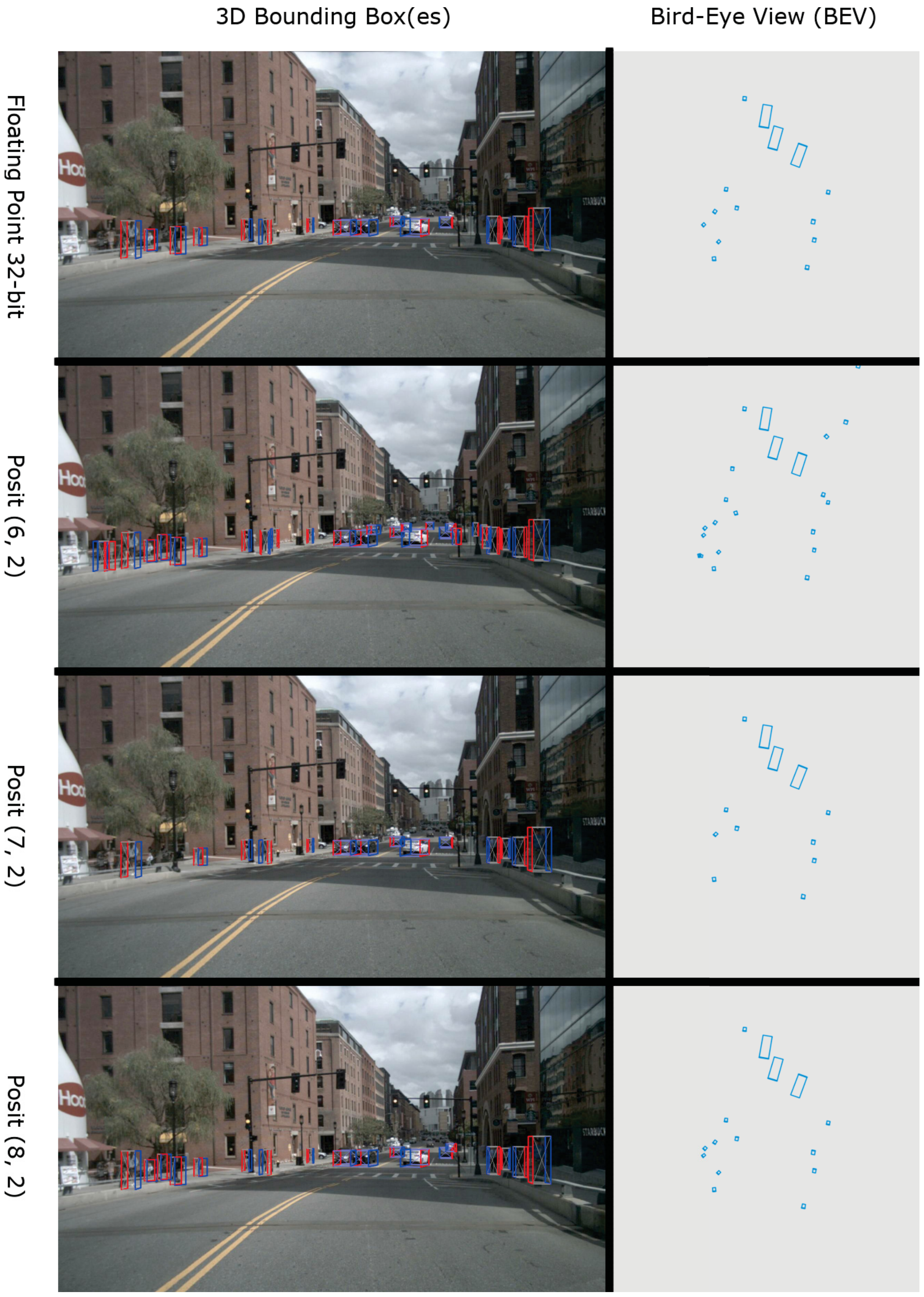
TABLE 16 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2), FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).

Format	mAP (-)	mATE (m)	mASE (-)	mAOE (rad)	mAVE (m/s)	mAAE (-)	NDS (-)
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (2, 2)	0.000	1.065	1.000	1.044	0.891	0.875	0.023
Posit (3, 2)	0.207	0.841	0.864	0.678	1.299	0.309	0.234
Posit (4, 2)	0.276	0.762	0.684	0.656	1.192	0.300	0.298
Posit (5, 2)	0.245	0.795	0.561	0.641	1.200	0.389	0.284
Posit (6, 2)	0.296	0.710	0.472	0.623	0.780	0.306	0.359
Posit (7, 2)	0.315	0.659	0.477	0.597	0.772	0.299	0.377
Posit (8, 2)	0.313	0.664	0.454	0.600	0.758	0.306	0.378
Posit (16, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

Additionally, FIGURE 33 presents the 3D bounding box predictions and bird's-eye views produced by the Posit-quantized CenterFusion model on a sample frame from the NuScenes minival dataset, when only the primary regression heads are quantized. These visualizations include Posit precision levels of 6, 7, and 8 bits, alongside the 32-bit floating-point baseline for comparison.

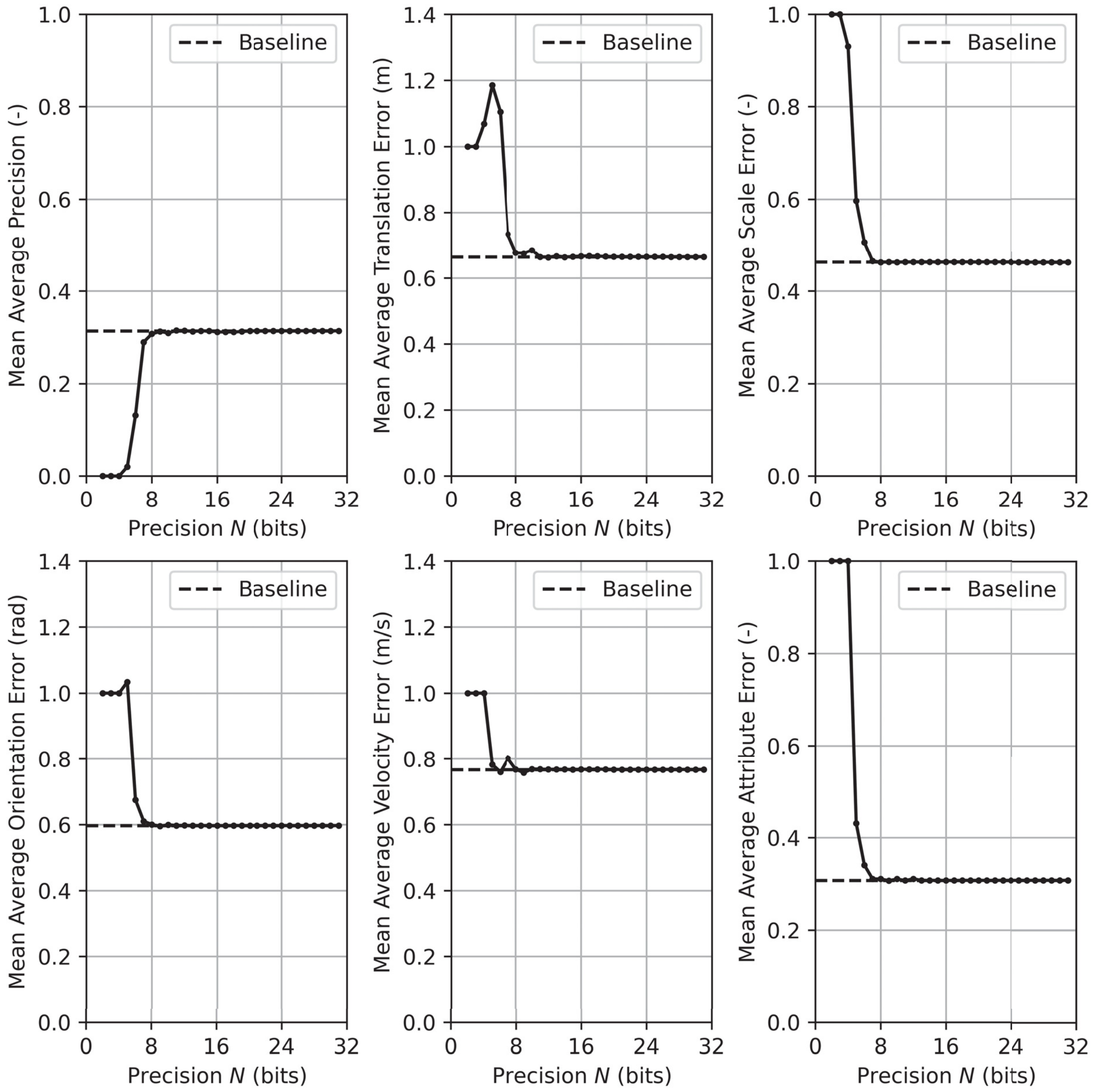
FIGURE 33 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

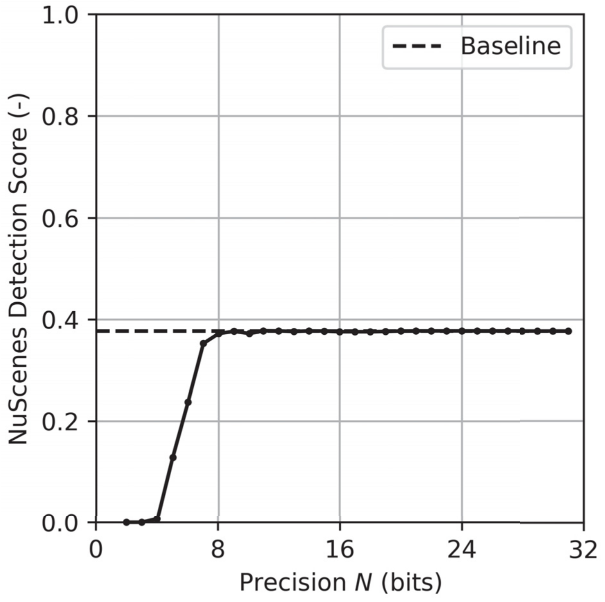
Similar curves can be seen in FIGURE 34 and FIGURE 35, corresponding to the sole quantization of CenterFusion's secondary regression heads, and whose specific values for each bit width can be read from TABLE 17. The corresponding degradations to the mAP and NDS for the Posit (7, 2) quantization is similar to those obtained for joint quantization of both regression heads, reaching 2.5% on the mAP and 2.5% on the NDS. This is the first indicative that the secondary regression heads have more impact on the quantization induced degradations to the metrics than the primary regression heads.

FIGURE 34 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

FIGURE 35 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2) FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

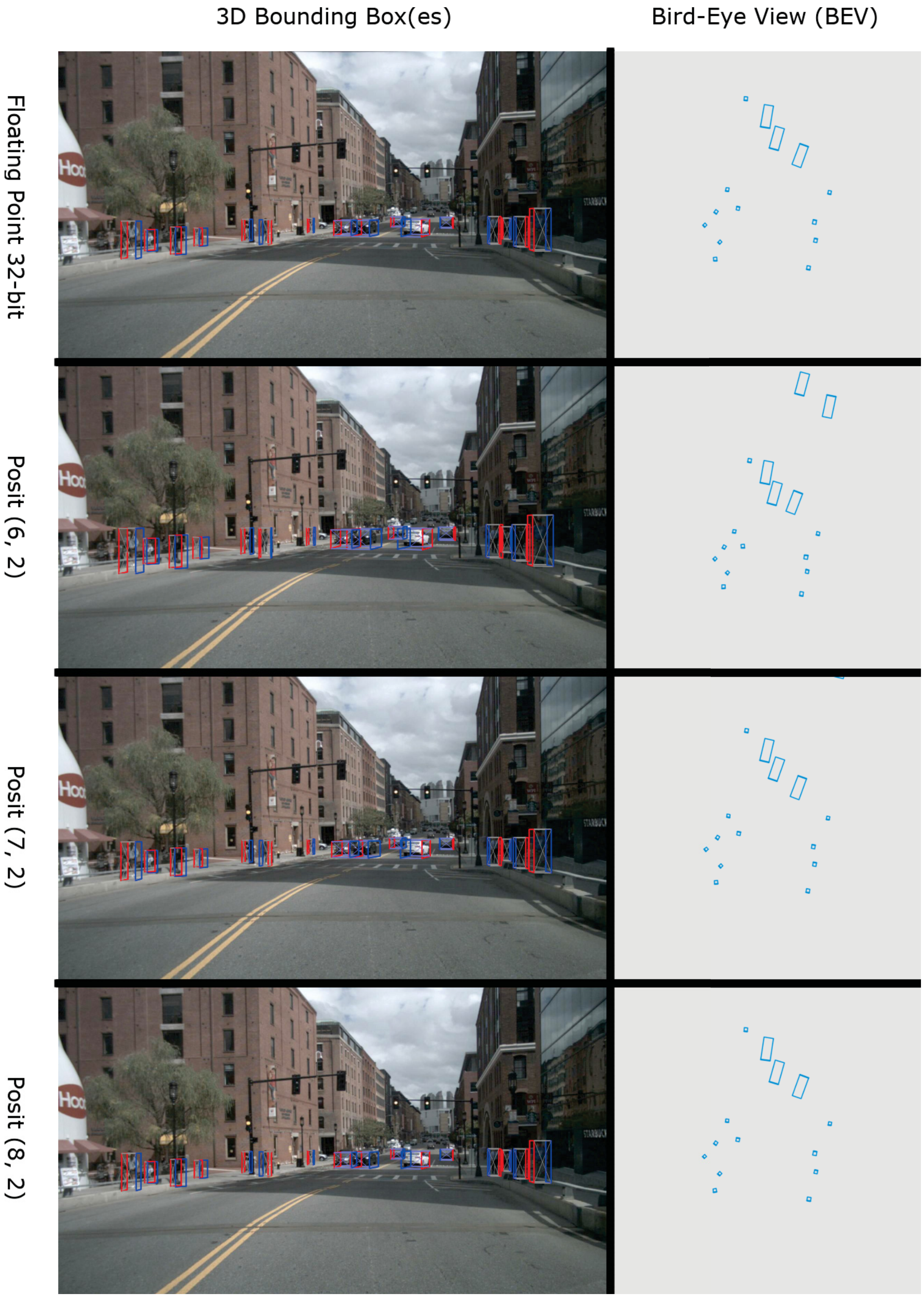
TABLE 17 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2), FROM QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).

Format	**mAP (-)**	**mATE (m)**	**mASE (-)**	**mAOE (rad)**	**mAVE (m/s)**	**mAAE (-)**	**NDS (-)**
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (2, 2)	0.000	1.000	1.000	1.000	1.000	1.000	0.000
Posit (3, 2)	0.000	1.000	1.000	1.000	1.000	1.000	0.000
Posit (4, 2)	0.000	1.068	0.931	1.000	1.000	1.000	0.007
Posit (5, 2)	0.020	1.186	0.597	1.034	0.782	0.431	0.129
Posit (6, 2)	0.132	1.104	0.506	0.675	0.759	0.341	0.238
Posit (7, 2)	0.289	0.734	0.465	0.610	0.802	0.311	0.352
Posit (8, 2)	0.307	0.677	0.462	0.600	0.767	0.311	0.372
Posit (16, 2)	0.311	0.666	0.463	0.596	0.767	0.308	0.376
Posit (32, 2)	0.314	0.664	0.463	0.597	0.767	0.308	0.377

SOURCE: The Author.

Again, FIGURE 36 shows the 3D bounding box predictions and bird's-eye views generated by the Posit-quantized CenterFusion model on a sample frame from the NuScenes minival dataset, with only the secondary regression heads quantized. These visualizations compare Posit precision levels of 6, 7, and 8 bits against the 32-bit floating-point baseline.

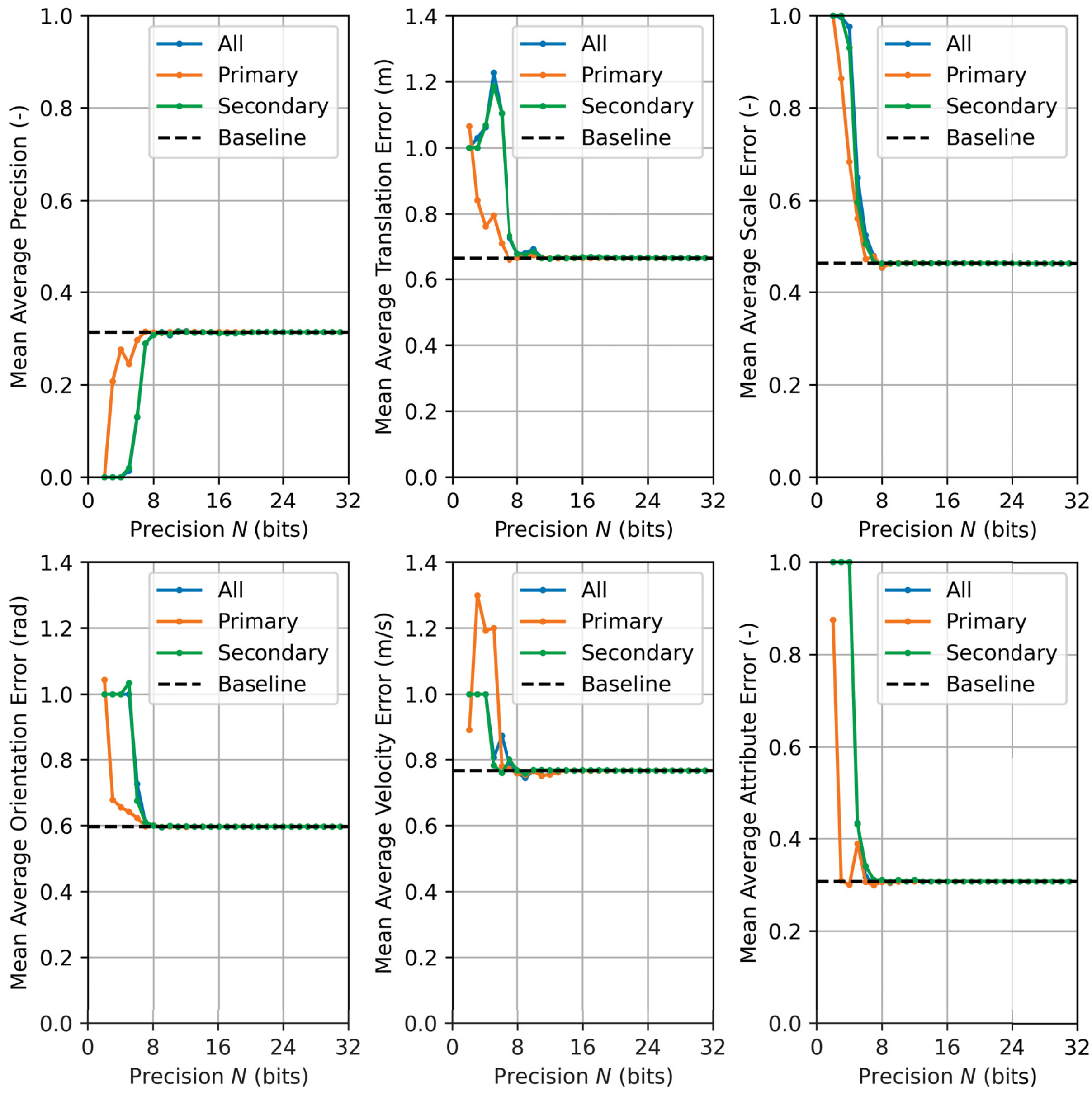
FIGURE 36 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

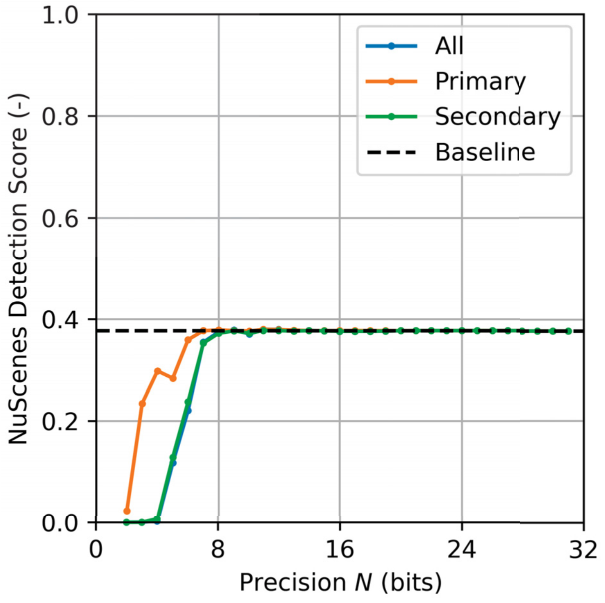
Additionally, the same curves discussed earlier are collectively presented in FIGURE 37 and FIGURE 38 to facilitate a comparison of the effects of quantization on each regression head individually. A notable finding from these figures is that the behavior of the metrics is closely linked to the secondary regression heads in CenterFusion. This relationship is confirmed by analyzing the error curves when each regression head is quantized separately. The graphs in FIGURE 37 and the NDS curve in FIGURE 38 show that the metric patterns observed when only the secondary regression heads are quantized closely resemble those seen when both regression heads undergo quantization simultaneously.

FIGURE 37 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT (N, 2) USING POSIT-AS-STORAGE (PAS).



SOURCE: The Author.

FIGURE 38 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT (N, 2) USING POSIT-AS-STORAGE (PAS).

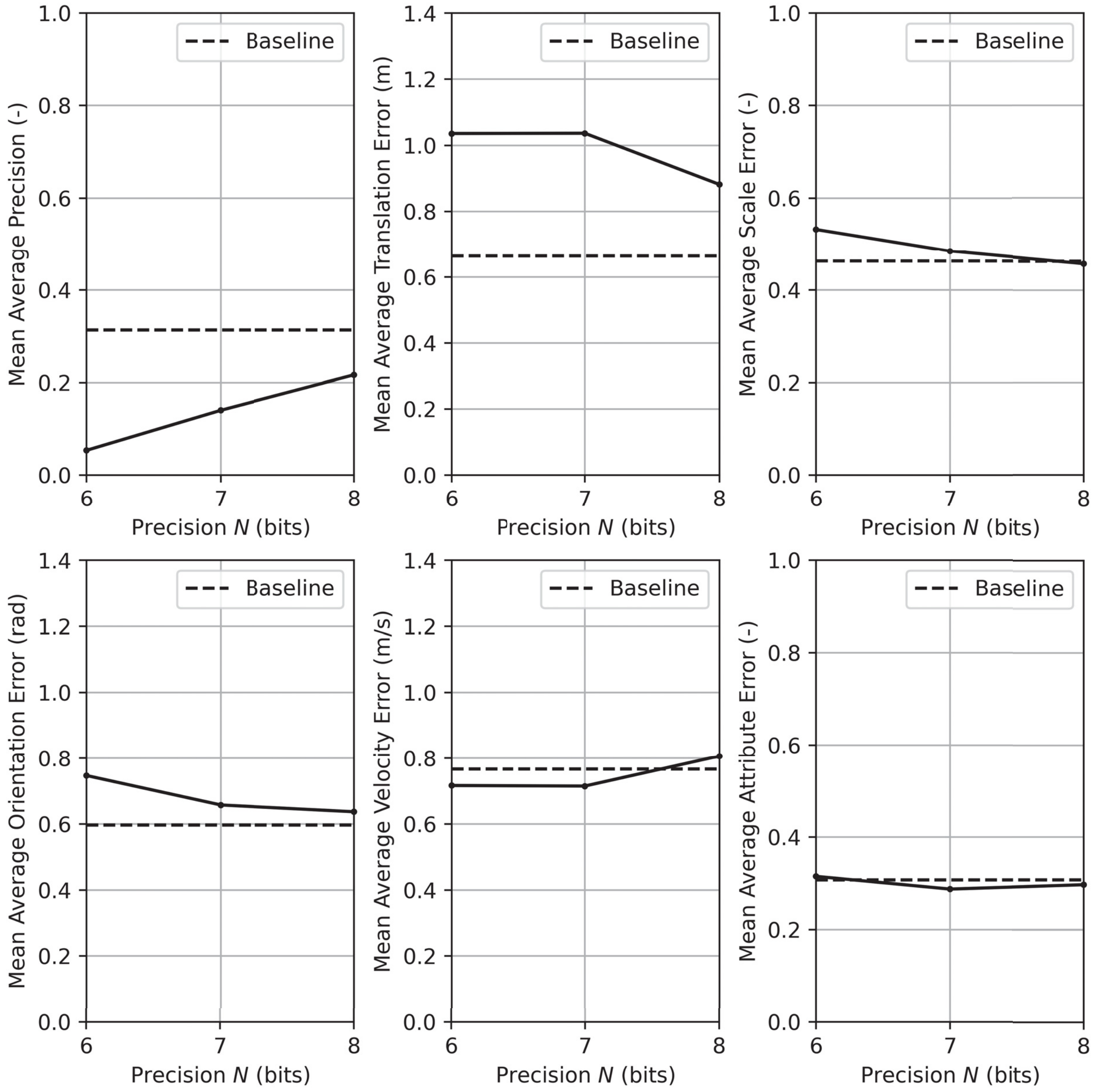


SOURCE: The Author.

5.2.2 Posit-as-Arithmetic (PaA)

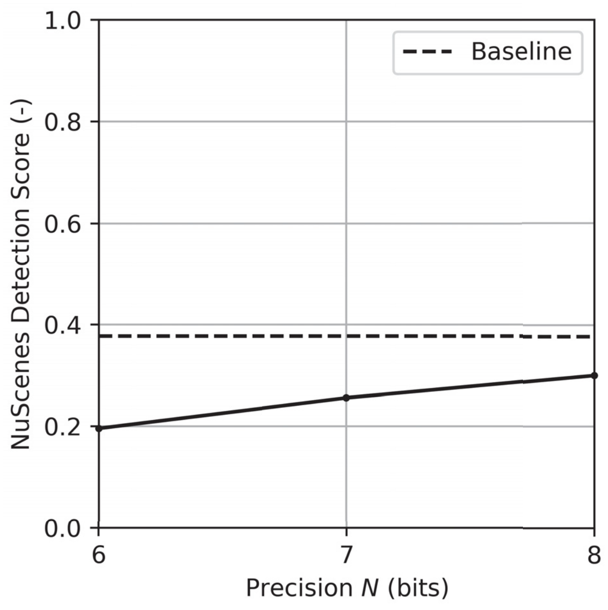
The metrics for PaA can be visualized in FIGURE 39 and FIGURE 40, where Posits (N, 2) with varying bit widths are used for joint quantization of CenterFusion's regression heads through Posits4TorcHA on the FPGA. For this strategy, the influence of both the Posit quantization of network parameters and Posit arithmetic on the FPGA is evident.

FIGURE 39 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

FIGURE 40 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

Specific values for each bit width are listed in TABLE 18. As expected, Posit (7, 2) quantization shows greater degradation when compared to 8-bit Posits, of up to 17.7% in mAP and 12.1% in NDS, as per TABLE 18. A similar observation is made for Posit (8, 2) quantization, which, when compared to its equivalent using PaS, shows greater degradation of up to 9.8% in mAP and 7.7% in NDS.

TABLE 18 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2), FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).

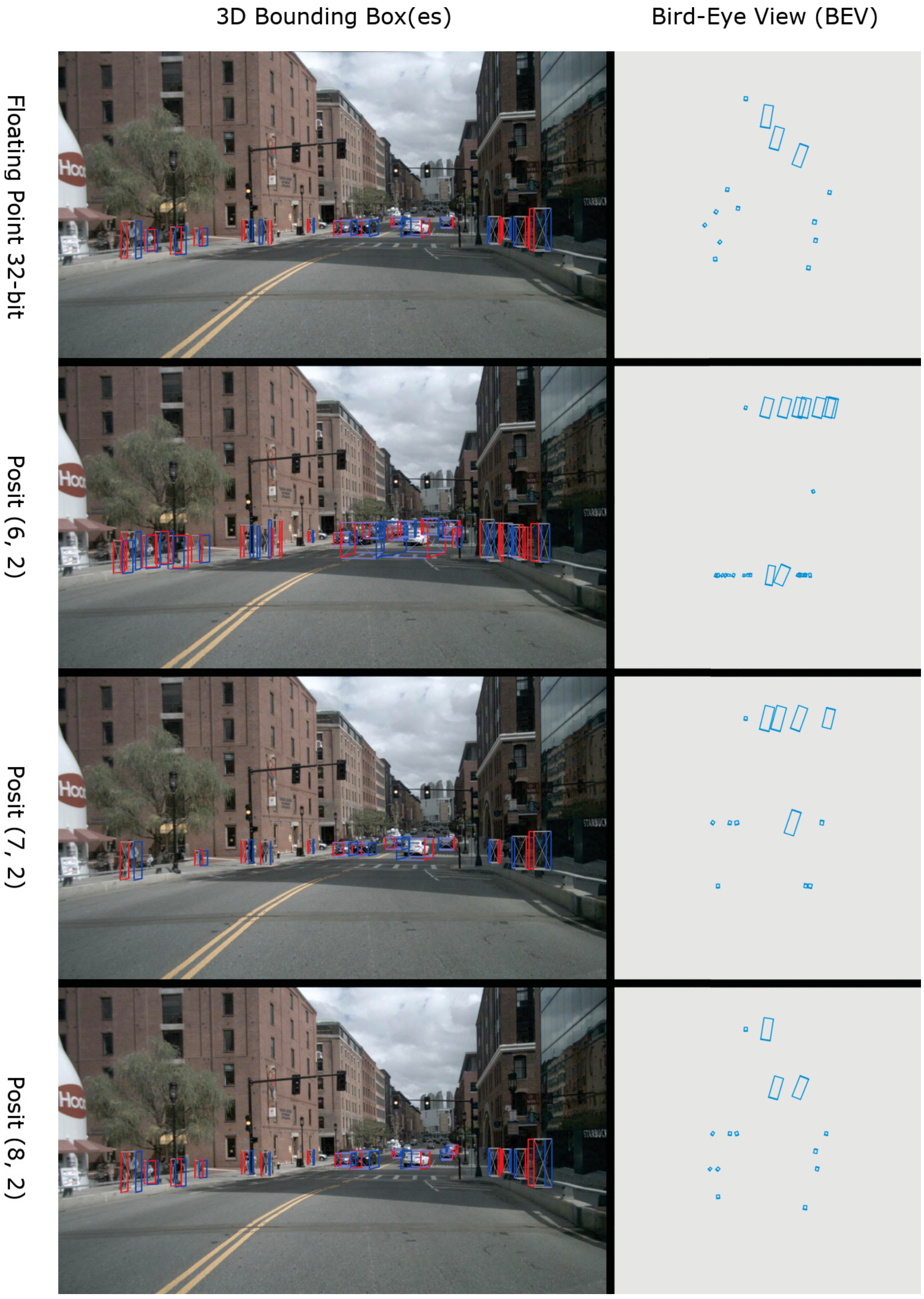
Format	**mAP (-)**	**mATE (m)**	**mASE (-)**	**mAOE (rad)**	**mAVE (m/s)**	**mAAE (-)**	**NDS (-)**
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (6, 2)	0.053	1.036	0.532	0.746	0.716	0.315	0.196
Posit (7, 2)	0.140	1.037	0.485	0.657	0.714	0.287	0.256
Posit (8, 2)	0.216	0.882	0.457	0.636	0.806	0.297	0.300

SOURCE: The Author.

Nevertheless, these results corroborate previous studies where similar configurations were successfully used for inference after quantizing other NN models, such as in (Kumar; Gupta, 2023; Edavoor et al., 2023; Glint et al., 2023; Zolfagharinejad et al., 2022; Immaneni et al., 2022; Langroudi et al., 2018; Cococcioni et al., 2020a,b; Langroudi et al., 2020), achieving similar metric degradations as seen here. However, it is important to note that direct comparisons are difficult due to differences in models and tasks involved in each work.

Furthermore, FIGURE 41 displays the 3D bounding box predictions and bird's-eye views generated by the Posit-quantized CenterFusion model on a sample frame from the NuScenes minival dataset, with both regression heads quantized. These visualizations show Posit precision levels of 6, 7, and 8 bits, compared alongside the 32-bit floating-point baseline.

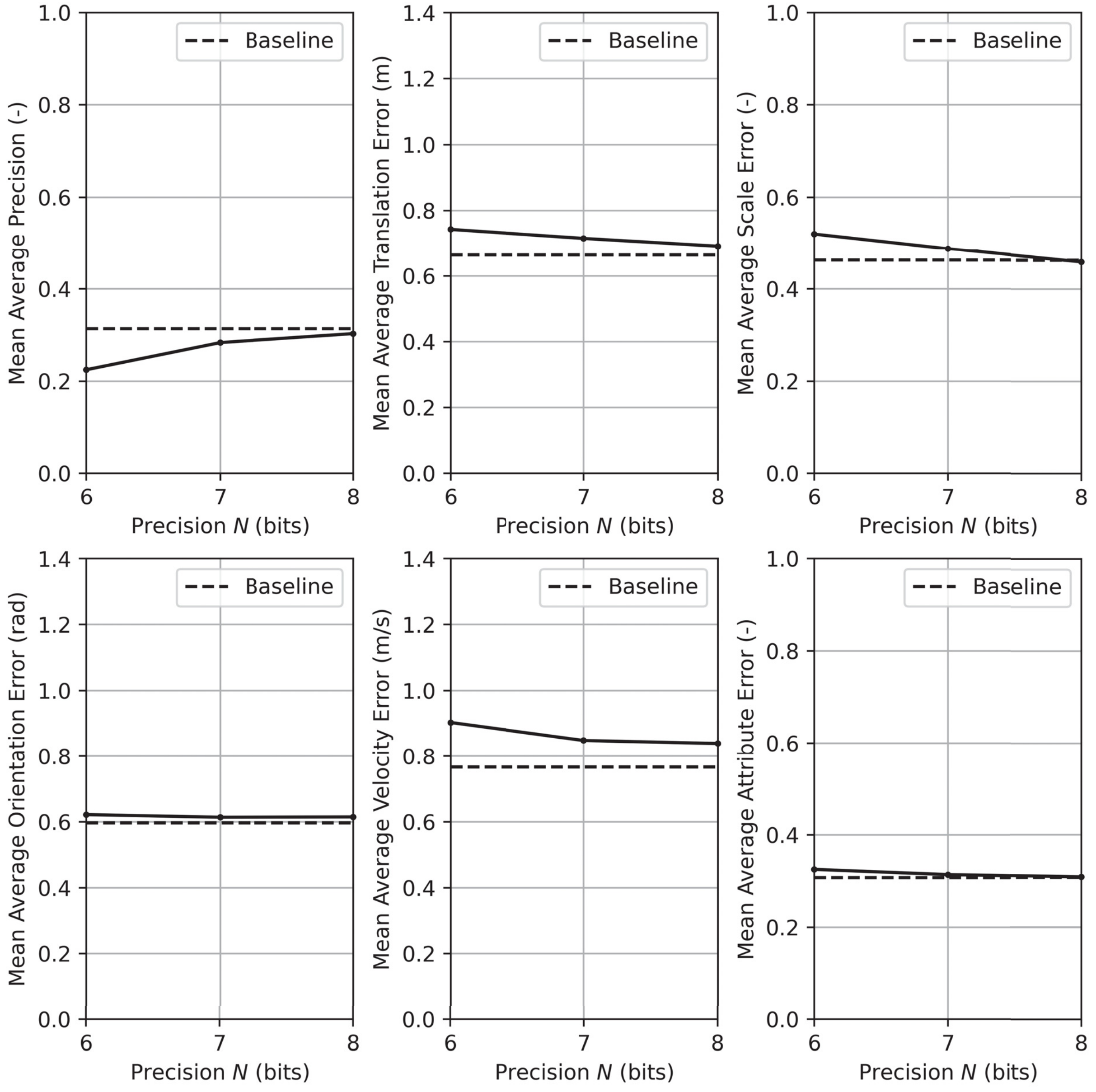
FIGURE 41 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF BOTH REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

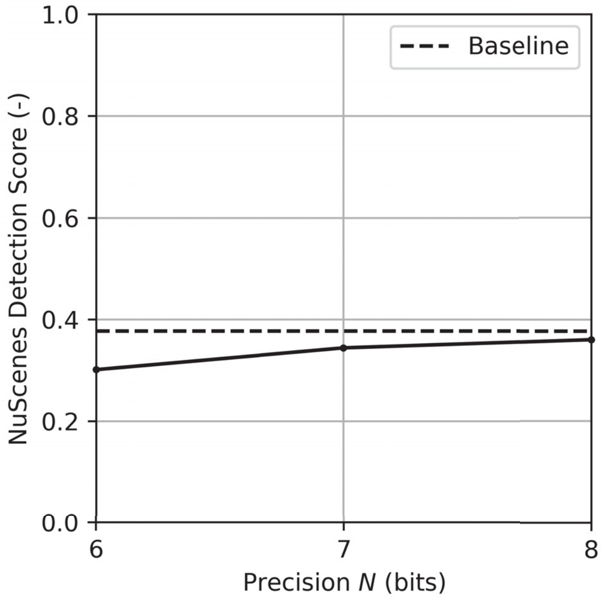
Similarly, FIGURE 42 and FIGURE 43 show the curves when only the primary regression heads are quantized using Posit (N, 2). Values for each bit width are in TABLE 19. In this case, a Posit (7, 2) quantization induces a degradation on the mAP and NDS of, respectively, 3.1% and 3.3%. For the Posit (8, 2) quantization scheme, the degradations to the mAP and NDS are of 1.1% and 1.7%, respectively.

FIGURE 42 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

FIGURE 43 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

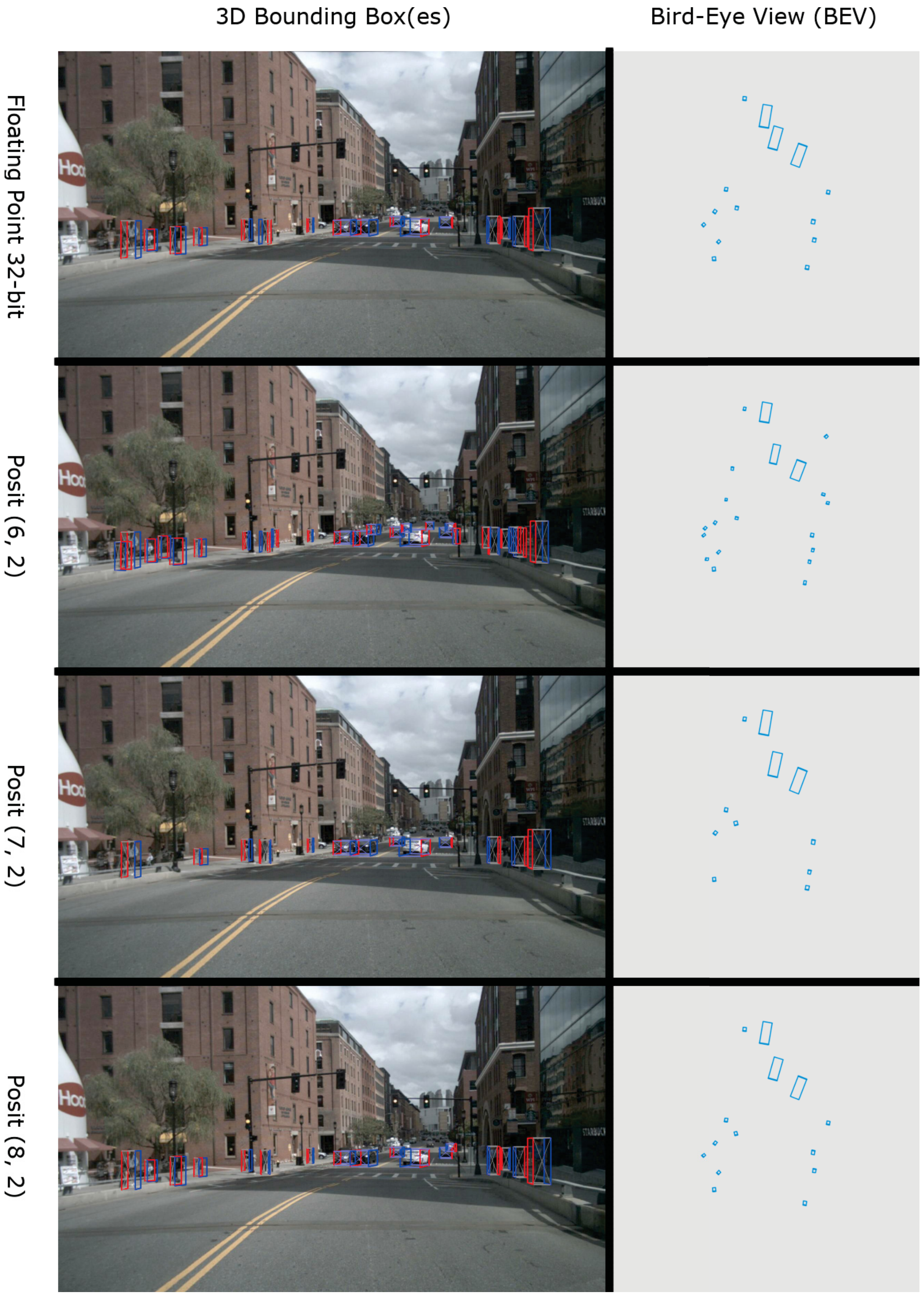
TABLE 19 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2), FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).

Format	**mAP (-)**	**mATE (m)**	**mASE (-)**	**mAOE (rad)**	**mAVE (m/s)**	**mAAE (-)**	**NDS (-)**
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (6, 2)	0.224	0.742	0.520	0.621	0.903	0.325	0.301
Posit (7, 2)	0.283	0.715	0.488	0.613	0.848	0.314	0.344
Posit (8, 2)	0.303	0.691	0.459	0.614	0.839	0.309	0.360

SOURCE: The Author.

In addition, FIGURE 44 presents the 3D bounding box predictions and bird's-eye views produced by the Posit-quantized CenterFusion model on a sample frame from the NuScenes minival dataset, with solely primary regression heads quantized. These visualizations compare Posit precision levels of 6, 7, and 8 bits alongside the 32-bit floating-point baseline.

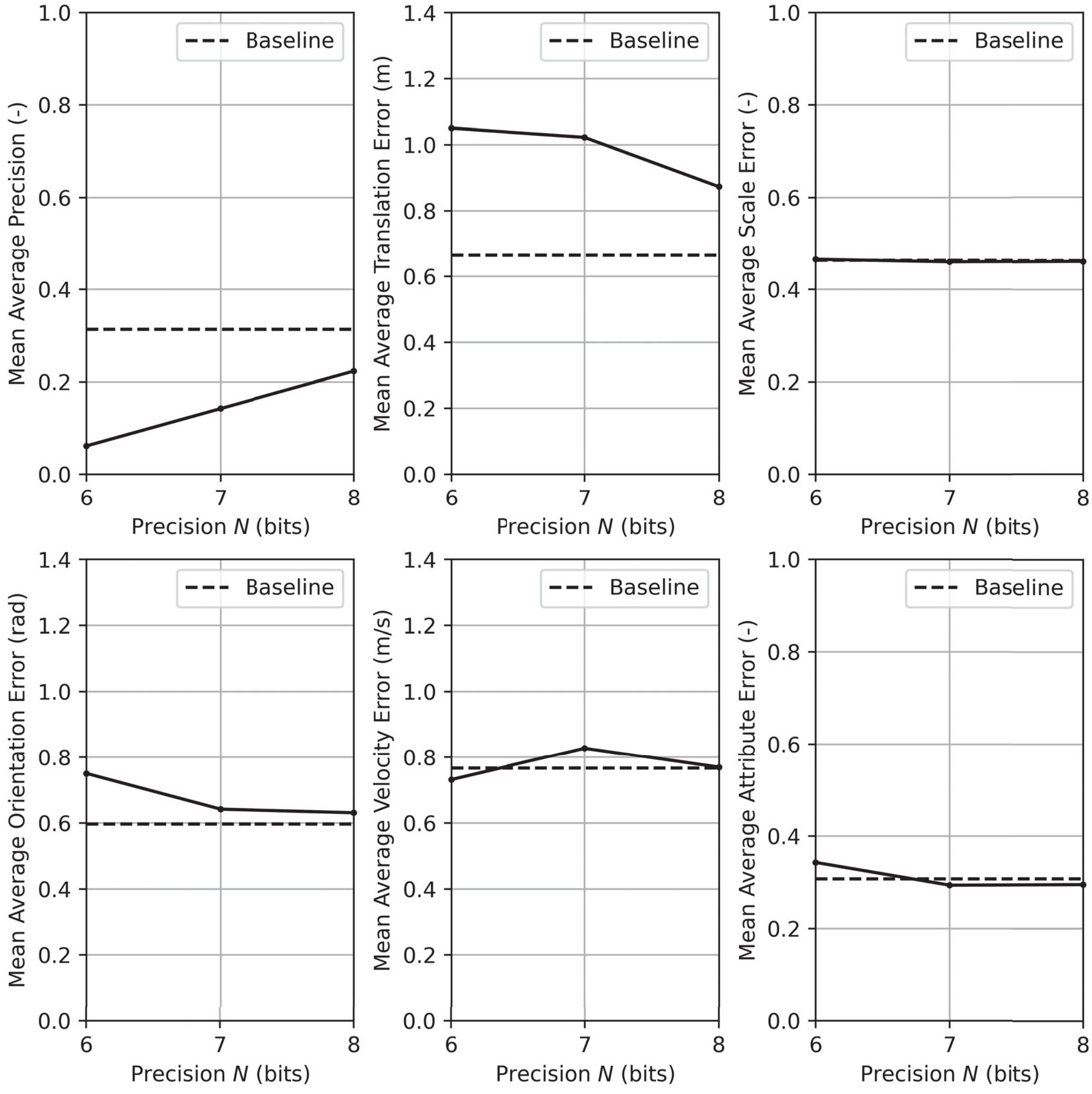
FIGURE 44 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF PRIMARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

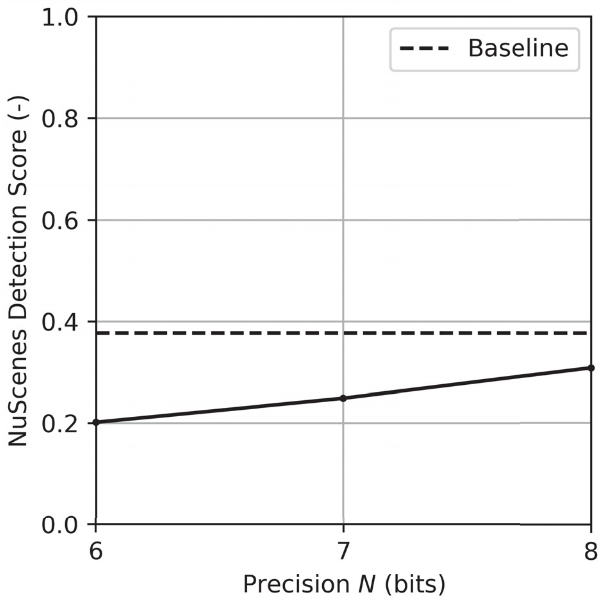
Moreover, FIGURE 45 and FIGURE 46 display the curves for the quantization of the secondary regression heads, whose values for each bit width are shown in TABLE 20. In this case, Posit (7, 2) quantization results in a decrease in mAP and NDS by 17.2% and 12.8%, respectively. With the Posit (8, 2) quantization scheme, the reductions in mAP and NDS are 9.1% and 6.8%, respectively.

FIGURE 45 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

FIGURE 46 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

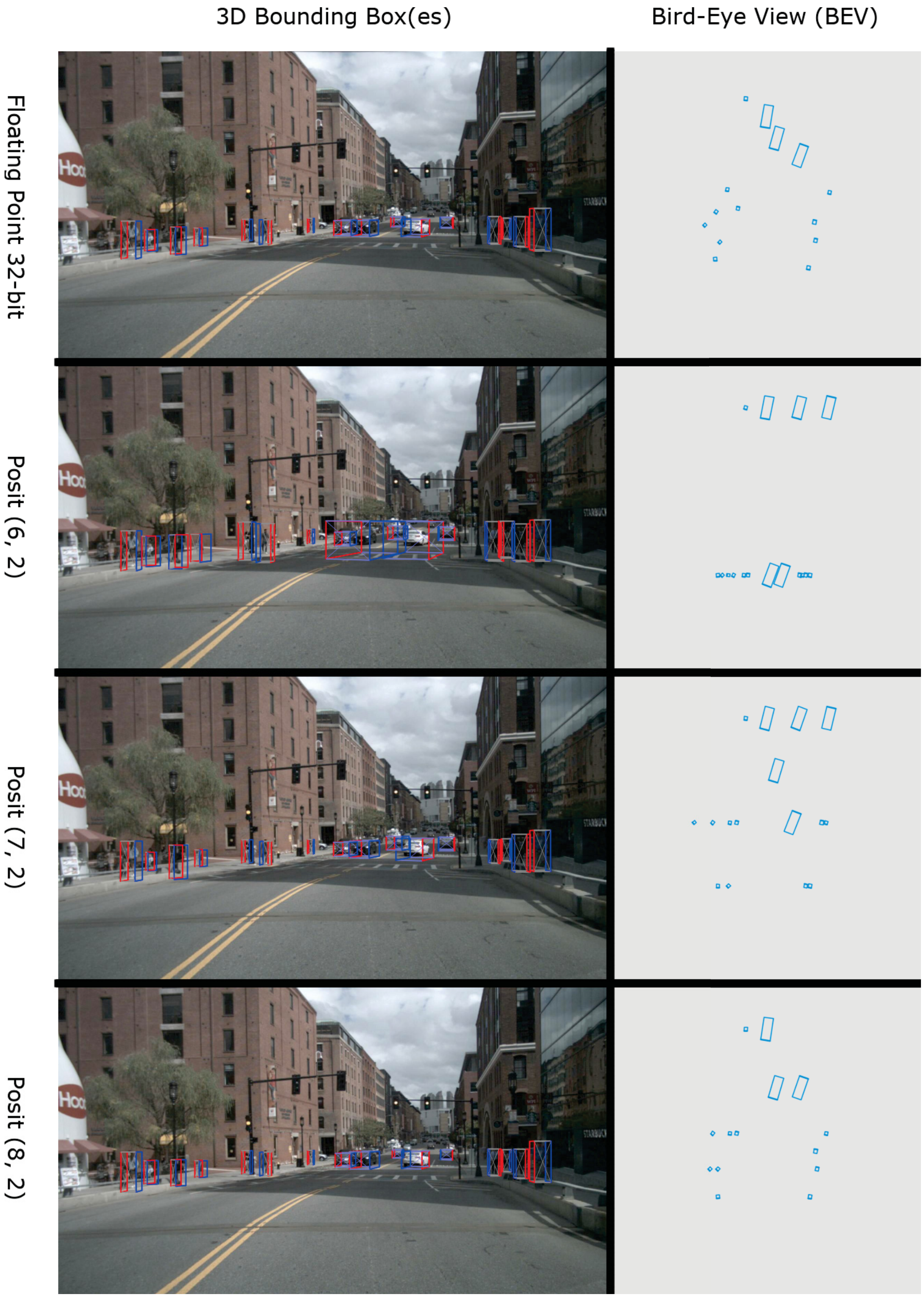
TABLE 20 – MEAN AVERAGE ERRORS AND NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2) FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).

Format	mAP (-)	mATE (m)	mASE (-)	mAOE (rad)	mAVE (m/s)	mAAE (-)	NDS (-)
Baseline	0.314	0.664	0.463	0.597	0.767	0.308	0.377
Posit (6, 2)	0.061	1.050	0.465	0.750	0.731	0.343	0.202
Posit (7, 2)	0.142	1.022	0.460	0.641	0.827	0.294	0.249
Posit (8, 2)	0.223	0.873	0.461	0.631	0.769	0.295	0.309

SOURCE: The Author.

Furthermore, FIGURE 47 illustrates the 3D bounding box predictions and bird's-eye views produced by the Posit-quantized CenterFusion model on a sample frame from the NuScenes minival dataset, with only secondary regression heads quantized. These visualizations include Posit precision levels of 6, 7, and 8 bits, alongside the 32-bit floating-point baseline for comparison.

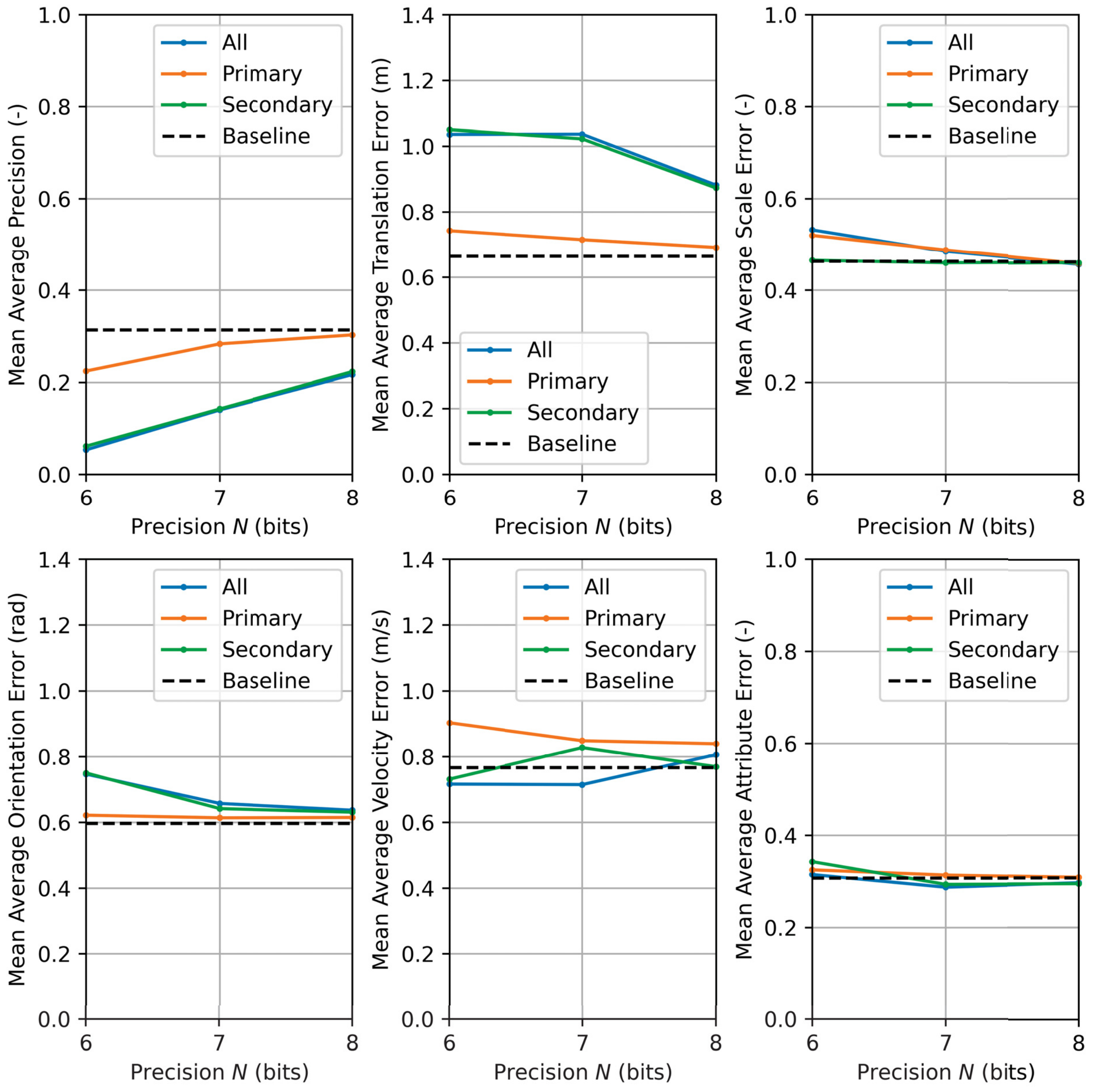
FIGURE 47 – 3D BOUNDING BOXES AND BIRD-EYE VIEWS OBTAINED FROM THE QUANTIZATION OF SECONDARY REGRESSION HEADS, USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

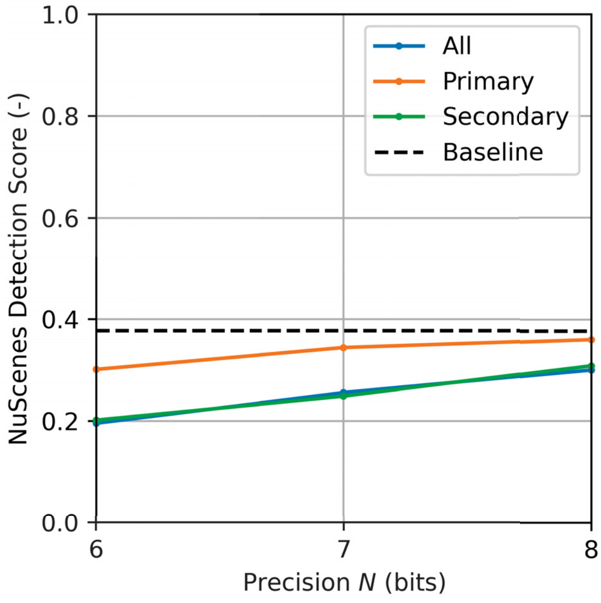
Finally, FIGURE 48 and FIGURE 49 together illustrate the effects of quantization on the metrics for each regression head. The behavior of the metrics under each quantization scheme resembles the patterns observed in the PaS strategy, where the metrics are closely tied to the secondary regression heads in CenterFusion. This alignment is validated by analyzing the error curves when quantizing each regression head individually. The graphs in FIGURE 48 and the NDS curve in FIGURE 49 show that the metric curves observed with quantization of only the secondary regression heads closely replicate those seen when both regression heads are quantized simultaneously.

FIGURE 48 – MEAN AVERAGE ERRORS OBTAINED WITH POSIT ({6, 7, 8}, 2) USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

FIGURE 49 – NUSCENES DETECTION SCORES (NDS) OBTAINED WITH POSIT ({6, 7, 8}, 2) USING POSIT-AS-ARITHMETIC (PAA).



SOURCE: The Author.

This suggests that these metrics are especially sensitive to the quantization of secondary regression heads, as previously noted, regardless of the quantization strategy applied, whether PaS or PaA. This phenomenon can be attributed to the different sensitivity to quantization between primary and secondary regression heads, as secondary regression heads are intended to regress properties over the combined camera and radar FMs, in contrast to primary regression heads, which regress object properties only from the FMs extracted from camera images. Thus, this affects the regressed value for velocity and translation metrics, the latter being related to the distance to the object, as these variables are closely related to radar measurements and are re-estimated through CenterFusion's secondary regression heads. This invariably leads to changes in the results obtained for the evaluation metrics corresponding to these properties. Nonetheless, secondary regression heads may be more affected due to having a larger number of convolutional layers and being closer to the network output. This finding is significant as it suggests an alternative quantization strategy where the primary and secondary regression heads are quantized using different Posit precisions. In this approach, the secondary regression heads are quantized with higher Posit precision than the primary regression heads, potentially mitigating quantization-induced degradations in the metrics. This strategy can also be applied to various parts of the model.

These results collectively demonstrate the effectiveness and potential applications of Posit quantization and hardware acceleration in DL models, particularly for AV perception systems. However, more research and optimizations are needed to fully realize the benefits of these techniques.

6 CONCLUSION AND FUTURE WORKS

The approach of Posit numbers is an innovation that aims to reduce the size of NNs and consequently improve their energy efficiency and resource utilization, especially in embedded systems, with minimal loss of inference accuracy. However, the literature on this representation format is still incipient, and more in-depth studies are needed, especially in the context of AVs and perception systems that use sensor fusion.

In this study, the functionality of Posits4Torch for quantization and inference using Posit numbers was presented and validated for a radar and camera middle-fusion method called CenterFusion, with applications in AV, especially in their perception systems that use AI applied to CV for 3D object detection and classification, as well as Posits4TorcHA for hardware-accelerated inference using the AMD Kria KV260 Vision Starter Kit. Functionallity validation results have demonstrated the potential of Posit arithmetic, particularly with Posit (8, 2), to maintain inference accuracy across different computation backends, including CPU-emulated and FPGA-based backends. Performance evaluation revealed a significant reduction in inference time with the FPGA-based Posit model, achieving an improvement of nearly 1000 times compared to the CPU-emulated Posit model, although it remained 300 times slower than the GPU-based floating-point model. This performance enhancement highlights the potential of FPGA-based Posit arithmetic for high-speed applications.

The case study on the CenterFusion model for AVs' perception systems further highlighted the applicability of Posit quantization in DL-based sensor fusion methods. Our findings indicate that an 8-bit Posit precision is sufficient to maintain comparable performance to that of the unquantized 32-bit floating-point model in key evaluation metrics, particularly when applied to secondary regression heads, when using the PaS strategy and was comparable to some fixed-point and floating-point quantizations from the literature when using the PaA strategy on an FPGA. However, we observed that certain metrics, such as mean Average Velocity Error (mAVE) and mean Average Translation Error (mATE), are more susceptible to quantization-induced degradation, emphasizing the need for careful consideration in model design and training.

Hardware utilization and power consumption analyses of the AMD Kria KV260 Vision Starter Kit provided insights into the trade-offs associated with different Posit precisions. Higher precisions, while offering better accuracy, resulted in a marginally higher power consumption, which is a critical factor for embedded systems and energy-constrained applications.

Finally, this work contributes to the advancement of research in AVs by introducing a new approach that optimizes the performance of NNs in perception and sensor fusion systems, crucial elements for the safety and efficiency of AVs. It does so through Posits4Torch and Posits4TorcHA, which offer a robust framework for Posit quantization and hardware acceleration, enabling efficient and accurate inference of DL models. The significant reduction in inference time and the feasibility of low-precision Posit arithmetic to reasonably maintain model performance open new possibilities for deploying high-speed, energy-efficient AI solutions in real-world applications.

The development of Posits4Torch and Posits4TorcHA has established a solid foundation for the use of Posit arithmetic in DL libraries and hardware acceleration. Based on this, several key areas for future work have been identified to improve performance and applicability.

The plan is to extend the module coverage of Posits4Torch to include a broader range of PyTorch modules, providing a comprehensive set of tools for NN architectures. Additionally, implementing a deformable convolution module for DCNs will enhance flexibility and efficiency in applications that require spatial adaptation, such as CenterFusion.

Also, the experimentation with more efficient Posit libraries, alternatives to the current SoftPosit library, such as the Universal Numbers Library (Omtzigt; Quinlan, 2023), is planned. The development of a dedicated DL framework other than Deep PeNSieve for software-emulated Posit numbers as a backend on CPU will also be a priority to optimize performance and control.

Investigating alternative communication protocols to HTTP aims to enhance remote hardware acceleration, reducing latency and improving throughput. There are also plans to experiment with different Posit MAC unit models from the literature and design a dedicated custom model for Posits4TorcHA, concurrently with the development of an optimized design for the 2D Posit MAC unit array, possibly using systolic arrays, aiming to increase overall performance. Transitioning from the overlay-style to the dataflow-style such as in hardware NNs similar to FINN (Umuroglu et al., 2017) will further improve performance through efficient data handling and greater parallelism.

Implementing hardware acceleration for a broader set of activation functions in Posits4TorcHA will address computational bottlenecks and speed up NN inference as well.

A case study on the posit quantization and inference of the DLA-34 backbone of CenterFusion will provide more insights into the benefits and challenges of the Posit arithmetic. Comparative analysis with 8-bit integer and fixed-point quantizations will further highlight the strengths and limitations of the PNS in terms of precision and resource efficiency.

Finally, comprehensive coding standards will be established to ensure consistency, maintainability, and high-quality code for the Posits4Torch and Posits4TorcHA projects, guiding future development and facilitating collaboration within the research community.

By addressing these areas, the aim is to solidify Posits4Torch and Posits4TorcHA as robust platforms, improving performance and expanding the applicability of Posit-based computations across various domains.

REFERENCES

AMD. **PYNQ**. Accessed in: 02/07/2024. 2016. Available at: https://www.pynq.io/. Cited on pages 52, 53.

BELABED, T.; COUTINHO, M. G. F.; FERNANDES, M. A.; SAKUYAMA, C. V.; SOUANI, C. User driven FPGA-based design automated framework of deep neural networks for low-power low-cost edge computing. **IEEE Access**, IEEE, v. 9, p. 89162–89180, 2021. Cited on page 23.

CAESAR, H.; BANKITI, V.; LANG, A. H.; VORA, S.; LIONG, V. E.; XU, Q.; KRISHNAN, A.; PAN, Y.; BALDAN, G.; BEIJBOM, O. nuScenes: A multimodal dataset for autonomous driving. In: CVPR. [S.l.: s.n.], 2020. Cited on page 19.

CIOCIRLAN, S.; LOGHIN, D.; RAMAPANTULU, L.; TĂPUŞ, N.; TEO, Y. The Accuracy and Efficiency of Posit Arithmetic. In: PROCEEDINGS - IEEE International Conference on Computer Design: VLSI in Computers and Processors. [S.l.: s.n.], 2021. 2021-Octob, p. 83–87. ISBN 9781665432191. DOI: 10.1109/ICCD53106.2021.00024. Available at: https://github.com/dloghin/posar%20https://github.com/si%EF%AC%81ve/freedom. Cited on page 46.

COCOCCIONI, M.; ROSSI, F.; RUFFALDI, E.; SAPONARA, S. A Novel Posit-based Fast Approximation of ELU Activation Function for Deep Neural Networks. In: PROCEEDINGS - 2020 IEEE International Conference on Smart Computing, SMARTCOMP 2020. [S.l.: s.n.], 2020. P. 244–246. ISBN 9781728169972. DOI: 10.1109/SMARTCOMP50058.2020.00053. Cited on pages 43, 46, 84.

________. Fast approximations of activation functions in deep neural networks when using posit arithmetic. **Sensors (Switzerland)**, v. 20, 5 2020. DOI: 10.3390/s20051515. Cited on pages 43, 46, 84.

COCOCCIONI, M.; ROSSI, F.; RUFFALDI, E.; SAPONARA, S.; DE DINECHIN, B. D. Novel Arithmetics in Deep Neural Networks Signal Processing for Autonomous Driving: Challenges and Opportunities. **IEEE Signal Processing Magazine**, v. 38, n. 1, p. 97–110, 2021. DOI: 10.1109/MSP.2020.2988436. Cited on page 37.

CRESPO, L.; TOMAS, P.; ROMA, N.; NEVES, N. Unified Posit/IEEE-754 Vector MAC Unit for Transprecision Computing. **IEEE Transactions on Circuits and Systems II:**

Express Briefs, v. 69, p. 2478–2482, 5 2022. DOI: 10.1109/TCSII.2022.3160191. Cited on pages 49, 50.

CRESPO, L.; TOMÁS, P.; ROMA, N.; NEVES, N. Trading Performance, Power, and Area on Low-Precision Posit MAC Units for CNN Training. In: IEEE. 2023 IEEE 35th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD). [S.l.: s.n.], 2023. Cited on pages 58, 59, 62.

DAI, J.; QI, H.; XIONG, Y.; LI, Y.; ZHANG, G.; HU, H.; WEI, Y. Deformable Convolutional Networks. In: 2017 IEEE International Conference on Computer Vision (ICCV). [S.l.: s.n.], 2017. P. 764–773. DOI: 10.1109/ICCV.2017.89. Cited on page 62.

EDAVOOR, P.; RAVEENDRAN, A.; SELVAKUMAR, D.; DESALPHINE, V.; SHANKAR, G.; RAUT, G. Design and Analysis of Posit Quire Processing Engine for Neural Network Applications. In: PROCEEDINGS of the IEEE International Conference on VLSI Design. [S.l.: s.n.], 2023. 2023-Janua, p. 252–257. ISBN 9798350346787. DOI: 10.1109/VLSID57277.2023.00059. Cited on pages 43, 46, 84.

ESSAM, M.; SHALABY, A.; TAHER, M. Design and Implementation of Low Power Posit Arithmetic Unit for Efficient Hardware Accelerators. In: PROCEEDINGS of the 10th International Japan-Africa Conference on Electronics, Communications, and Computations, JAC-ECC 2022. [S.l.: s.n.], 2022. P. 68–71. ISBN 9781665464635. DOI: 10.1109/JAC-ECC56395.2022.10043893. Cited on page 46.

GLINT, T.; PRASAD, K.; DAGLI, J.; GANDHI, K.; GUPTA, A.; PATEL, V.; SHAH, N.; MEKIE, J. Hardware-Software Codesign of DNN Accelerators Using Approximate Posit Multipliers. In: PROCEEDINGS of the Asia and South Pacific Design Automation Conference, ASP-DAC. [S.l.: s.n.], 2023. P. 469–474. ISBN 9781450397834. DOI: 10.1145/3566097.3567866. Cited on pages 43, 45, 46, 84.

GOHIL, V.; WALIA, S.; MEKIE, J.; AWASTHI, M. Fixed-Posit: A Floating-Point Representation for Error-Resilient Applications. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 68, p. 3341–3345, 10 2021. DOI: 10.1109/TCSII.2021.3072217. Cited on pages 43, 45, 46.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. **Deep Learning**. [S.l.]: MIT Press, 2016. http://www.deeplearningbook.org. Cited on pages 24, 34.

GUSTAFSON, J.; YONEMOTO, I. Beating floating point at its own game: Posit arithmetic. **Supercomputing Frontiers and Innovations**, v. 4, p. 71–86, 2 2017. DOI: 10.14529/jsfi170206. Cited on pages 19, 36, 41.

HENNESSY, J. L.; PATTERSON, D. A. **Computer Architecture: A Quantitative Approach**. 6th. [S.l.]: Morgan Kaufmann, 2017. Cited on page 36.

IMMANENI, A.; ULLAH, S.; NAMBI, S.; SAHOO, S.; KUMAR, A. PosAx-O: Exploring Operator-level Approximations for Posit Arithmetic in Embedded AI/ML. In: PROCEEDINGS - 2022 25th Euromicro Conference on Digital System Design, DSD 2022. [S.l.: s.n.], 2022. P. 214–223. ISBN 9781665474047. DOI: 10.1109/DSD57027.2022.00037. Cited on pages 46, 84.

JAISWAL, M. K.; SO, H. K.-H. PACoGen: A Hardware Posit Arithmetic Core Generator. **IEEE Access**, v. 7, p. 74586–74601, 2019. DOI: 10.1109/ACCESS.2019.2920936. Cited on page 47.

KANT, M.; THAKUR, R. Implementation and Performance Improvement of POSIT Multiplier for Advance DSP Applications. In: PROCEEDINGS of the 5th International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud), I-SMAC 2021. [S.l.: s.n.], 2021. P. 1730–1736. ISBN 9781665426428. DOI: 10.1109/I-SMAC52330.2021.9640999. Cited on page 46.

KUMAR, N.; GUPTA, V. A Posit based Handwritten Digits Recognition System. **International Journal of Computing and Digital Systems**, v. 13, p. 1217–1225, 1 2023. DOI: 10.12785/ijcds/130199. Cited on pages 43, 46, 84.

LANGROUDI, H.; KARIA, V.; CARMICHAEL, Z.; ZYARAH, A.; PANDIT, T.; GUSTAFSON, J.; KUDITHIPUDI, D. Alps: Adaptive quantization of deep neural networks with generalized posits. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. [S.l.: s.n.], 2021. P. 3094–3103. ISBN 9781665448994. DOI: 10.1109/CVPRW53098.2021.00346. Cited on pages 45, 46.

LANGROUDI, H.; KARIA, V.; GUSTAFSON, J.; KUDITHIPUDI, D. Adaptive posit: Parameter aware numerical format for deep learning inference on the edge. In: IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops. [S.l.: s.n.], 2020. 2020-June, p. 3123–3131. ISBN 9781728193601. DOI: 10.1109/CVPRW50498.2020.00371. Cited on pages 43, 45, 55, 84.

LANGROUDI, S.; PANDIT, T.; KUDITHIPUDI, D. Deep Learning Inference on Embedded Devices: Fixed-Point vs Posit. In: PROCEEDINGS - 1st Workshop on Energy Efficient Machine Learning and Cognitive Computing for Embedded Applications, EMC2 2018. [S.l.: s.n.], 2018. P. 19–23. ISBN 9781538673676. DOI: 10.1109/EMC2.2018.00012. Cited on pages 43, 45, 55, 84.

LEE, J.; MUKHANOV, L.; MOLAHOSSEINI, A.; MINHAS, U.; HUA, Y.; RINCON, J. M. D.; DICHEV, K.; HONG, C.-H.; VANDIERENDONCK, H. Resource-Efficient Convolutional Networks: A Survey on Model-, Arithmetic-, and Implementation-Level Techniques. **ACM Computing Surveys**, v. 55, 13 s 2023. DOI: 10.1145/3587095. Cited on pages 26, 29.

LEONG, C. **SoftPosit**. Accessed in: 03/01/2024. 2018. Available at: https://gitlab.com/cerlane/SoftPosit. Cited on pages 44, 52, 54.

LI, Q.; FANG, C.; WANG, Z. PDPU: An Open-Source Posit Dot-Product Unit for Deep Learning Applications. In: PROCEEDINGS - IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2023. 2023-May. ISBN 9781665451093. DOI: 10.1109/ISCAS46773.2023.10182007. Cited on page 50.

LU, J.; LU, S.; WANG, Z.; FANG, C.; LIN, J.; WANG, Z.; DU, L. Training Deep Neural Networks Using Posit Number System. In: INTERNATIONAL System on Chip Conference. [S.l.: s.n.], 2019. 2019-Septe, p. 62–67. ISBN 9781728134826. DOI: 10.1109/SOCC46988.2019.1570558530. Cited on pages 43, 45, 50, 55.

MISHRA, S.; TIWARI, A.; SHEKHAWAT, H.; GUHA, P.; TRIVEDI, G.; JAN, P.; NEMEC, Z. Comparison of Floating-point Representations for the Efficient Implementation of Machine Learning Algorithms. In: 2022 32nd International Conference Radioelektronika, RADIOELEKTRONIKA 2022 - Proceedings. [S.l.: s.n.], 2022. ISBN 9781728186863. DOI: 10.1109/RADIOELEKTRONIKA54537.2022.9764927. Available at: https://github.com/manish-kj/Posit-HDL-%20Arithmetic. Cited on page 50.

MURILLO, R.; BARRIO, A. D.; BOTELLA, G. Customized posit adders and multipliers using the FloPoCo core generator. In: PROCEEDINGS - IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2020. 2020-Octob. ISBN 9781728133201. Cited on pages 46, 47.

________. Deep PeNSieve: A deep learning framework based on the posit number system. **Digital Signal Processing: A Review Journal**, v. 102, 2020. DOI:

10.1016/j.dsp.2020.102762. Available at:
https://github.com/RaulMurillo/deep-pensieve. Cited on pages 43, 45, 52.

MURILLO, R.; BARRIO, A. D.; BOTELLA, G.; KIM, M.; KIM, H.; BAGHERZADEH, N. PLAM: A Posit Logarithm-Approximate Multiplier. **IEEE Transactions on Emerging Topics in Computing**, v. 10, p. 2079–2085, 4 2022. DOI: 10.1109/TETC.2021.3109127. Cited on pages 43, 46, 47.

MURILLO, R.; BARRIO, A.; BOTELLA, G. The Effects of Numerical Precision In Scientific Applications. In: PROCEEDINGS of the 2022 Annual Modeling and Simulation Conference, ANNSIM 2022. [S.l.: s.n.], 2022. P. 152–163. ISBN 9781713852889. DOI: 10.23919/ANNSIM55834.2022.9859379. Cited on pages 43, 46.

MURILLO, R.; MALLASÉN, D.; BARRIO, A. D.; BOTELLA, G. Energy-Efficient MAC Units for Fused Posit Arithmetic. In: PROCEEDINGS - IEEE International Conference on Computer Design: VLSI in Computers and Processors. [S.l.: s.n.], 2021. 2021-Octob, p. 138–145. ISBN 9781665432191. DOI: 10.1109/ICCD53106.2021.00032. Cited on pages 43, 45–47, 50, 59.

NABATI, R.; QI, H. CenterFusion: Center-based Radar and Camera Fusion for 3D Object Detection. In: 2021 IEEE Winter Conference on Applications of Computer Vision (WACV). [S.l.: s.n.], 2021. P. 1526–1535. DOI: 10.1109/WACV48630.2021.00157. Cited on pages 19, 30, 31.

NAKAHARA, Y.; MASUDA, Y.; KIYAMA, M.; AMAGASAKI, M.; IIDA, M. A Posit Based Multiply-accumulate Unit with Small Quire Size for Deep Neural Networks. **IPSJ Transactions on System LSI Design Methodology**, v. 15, p. 16–19, 2022. DOI: 10.2197/ipsjtsldm.15.16. Available at:
https://github.com/nakahara457/%EF%AC%82oat-like-quire-posit. Cited on pages 45, 46, 50, 55, 59.

NAMBI, S.; ULLAH, S.; SAHOO, S. S.; LOHANA, A.; MERCHANT, F.; KUMAR, A. ExPAN(N)D: Exploring Posits for Efficient Artificial Neural Network Design in FPGA-Based Systems. **IEEE Access**, v. 9, p. 103691–103708, 2021. DOI: 10.1109/ACCESS.2021.3098730. Cited on pages 39, 40.

NEVES, N.; TOMAS, P.; ROMA, N. Reconfigurable stream-based tensor unit with variable-precision posit arithmetic. In: PROCEEDINGS of the International Conference on Application-Specific Systems, Architectures and Processors. [S.l.: s.n.], 2020.

2020-July, p. 149–156. ISBN 9781728171470. DOI: 10.1109/ASAP49362.2020.00033. Cited on pages 50, 51.

NORRIS, C.; KIM, S. An approximate and iterative posit multiplier architecture for FPGAs. In: PROCEEDINGS - IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2021. 2021-May. ISBN 9781728192017. DOI: 10.1109/ISCAS51556.2021.9401158. Cited on page 46.

OMTZIGT, E. T. L.; QUINLAN, J. Universal Numbers Library: Multi-format Variable Precision Arithmetic Library. **Journal of Open Source Software**, v. 8, n. 83, p. 5072, 2023. Cited on page 96.

PASZKE, A.; GROSS, S.; MASSA, F.; LERER, A.; BRADBURY, J.; CHANAN, G.; KILLEEN, T.; LIN, Z.; GIMELSHEIN, N.; ANTIGA, L. et al. Pytorch: An imperative style, high-performance deep learning library. **Advances in neural information processing systems**, v. 32, 2019. Cited on pages 44, 52.

PODOBAS, A.; MATSUOKA, S. Hardware implementation of POSITs and their application in FPGAs. In: PROCEEDINGS - 2018 IEEE 32nd International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2018. [S.l.: s.n.], 2018. P. 138–145. ISBN 9781538655559. DOI: 10.1109/IPDPSW.2018.00029. Cited on page 47.

RAPOSO, G.; TOMÁS, P.; ROMA, N. PositNN: Training deep neural networks with mixed low-precision posit. In: ICASSP, IEEE International Conference on Acoustics, Speech and Signal Processing - Proceedings. [S.l.: s.n.], 2021. 2021-June, p. 7908–7912. DOI: 10.1109/ICASSP39728.2021.9413919. Available at: https://github.com/hpc-ulisboa/posit-neuralnet. Cited on pages 43, 44, 46.

SHEKHAWAT, D.; JANGIR, A.; PANDEY, J. A Hardware Generator for Posit Arithmetic and its FPGA Prototyping. In: 2021 25th International Symposium on VLSI Design and Test, VDAT 2021. [S.l.: s.n.], 2021. ISBN 9781665419925. DOI: 10.1109/VDAT53777.2021.9601025. Cited on page 46.

TAMBE, T.; YANG, E.-Y.; WAN, Z.; DENG, Y.; REDDI, V. J.; RUSH, A.; BROOKS, D.; WEI, G.-Y. Algorithm-hardware co-design of adaptive floating-point encodings for resilient deep learning inference. In: PROCEEDINGS - Design Automation Conference. [S.l.: s.n.], 2020. 2020-July. ISBN 9781450367257. DOI: 10.1109/DAC18072.2020.9218516. Cited on page 46.

UGUEN, Y.; FORGET, L.; DINECHIN, F. D. Evaluating the hardware cost of the posit number system. In: PROCEEDINGS - 29th International Conference on Field-Programmable Logic and Applications, FPL 2019. [S.l.: s.n.], 2019. P. 106–113. ISBN 9781728148847. DOI: 10.1109/FPL.2019.00026. Cited on pages 46, 47.

UMUROGLU, Y.; FRASER, N. J.; GAMBARDELLA, G.; BLOTT, M.; LEONG, P.; JAHRE, M.; VISSERS, K. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. In: PROCEEDINGS of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. [S.l.]: ACM, 2017. (FPGA '17), p. 65–74. Cited on page 96.

VASILEV, I.; SLATER, D. **Python Deep Learning**. 2nd. Birmingham, UK: Packt Publishing, 2019. ISBN 978-1-78899-890-3. Cited on pages 22–24, 26, 28.

WALIA, S.; TEJ, B.; KABRA, A.; DEVNATH, J.; MEKIE, J. Fast and Low-Power Quantized Fixed Posit High-Accuracy DNN Implementation. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 30, p. 108–111, 1 2022. DOI: 10.1109/TVLSI.2021.3131609. Cited on pages 43, 46, 47.

WANG, Y.; DENG, D.; LIU, L.; WEI, S.; YIN, S. LPE: Logarithm Posit Processing Element for Energy-Efficient Edge-Device Training. In: 2021 IEEE 3rd International Conference on Artificial Intelligence Circuits and Systems, AICAS 2021. [S.l.: s.n.], 2021. ISBN 9781665419130. DOI: 10.1109/AICAS51828.2021.9458421. Cited on pages 43, 45, 49, 51.

YU, F.; WANG, D.; SHELHAMER, E.; DARRELL, T. Deep layer aggregation. In: PROCEEDINGS of the IEEE conference on computer vision and pattern recognition. [S.l.: s.n.], 2018. P. 2403–2412. Cited on page 30.

ZHANG, H.; HE, J.; KO, S.-B. Efficient posit multiply-accumulate unit generator for deep learning applications. In: PROCEEDINGS - IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2019. 2019-May. ISBN 9781728103976. DOI: 10.1109/ISCAS.2019.8702349. Cited on page 47.

ZHANG, H.; KO, S.-B. Design of Power Efficient Posit Multiplier. **IEEE Transactions on Circuits and Systems II: Express Briefs**, v. 67, p. 861–865, 5 2020. DOI: 10.1109/TCSII.2020.2980531. Cited on pages 46, 47.

ZHANG, H.; KO, S.-B. Efficient Approximate Posit Multipliers for Deep Learning Computation. **IEEE Journal on Emerging and Selected Topics in Circuits and Systems**, v. 13, p. 201–211, 1 2023. DOI: 10.1109/JETCAS.2022.3231642. Cited on pages 43, 46, 47.

________. Efficient multiple-precision posit multiplier. In: PROCEEDINGS - IEEE International Symposium on Circuits and Systems. [S.l.: s.n.], 2021. 2021-May. ISBN 9781728192017. DOI: 10.1109/ISCAS51556.2021.9401213. Cited on pages 46, 47.

ZHOU, X.; WANG, D.; KRÄHENBÜHL, P. **Objects as Points**. [S.l.: s.n.], 2019. arXiv: 1904.07850 [cs.CV]. Cited on page 30.

ZOLFAGHARINEJAD, M.; KAMAL, M.; AFZALI-KHUSHA, A.; PEDRAM, M. Posit Process Element for Using in Energy-Efficient DNN Accelerators. **IEEE Transactions on Very Large Scale Integration (VLSI) Systems**, v. 30, p. 844–848, 6 2022. DOI: 10.1109/TVLSI.2022.3165510. Cited on pages 43, 45, 46, 49, 50, 84.