MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

DESIGN ASPECTS OF MACHINE LEARNING ALGORITHMS FOR THE HARDWARE IMPLEMENTATION OF ADVANCED DRIVER ASSISTANCE SYSTEMS (A/DAS)

GÜNER TATAR

Ph.D. THESIS

Department of Electrical-Electronics Engineering

Electrical-Electronics Engineering

Thesis Supervisor

Asst. Prof. Salih BAYAR

Thesis Co-Supervisor

Asst. Prof. İhsan ÇİÇEK

ISTANBUL, 2024

MARMARA UNIVERSITY

INSTITUTE FOR GRADUATE STUDIES IN

PURE AND APPLIED SCIENCES

DESIGN ASPECTS OF MACHINE LEARNING ALGORITHMS FOR THE HARDWARE IMPLEMENTATION OF ADVANCED DRIVER ASSISTANCE SYSTEMS (A/DAS)

GÜNER TATAR
(725017006)

Ph.D. THESIS

Department of Electrical-Electronics Engineering

Electrical-Electronics Engineering

Thesis Supervisor

Asst. Prof. Salih BAYAR

Thesis Co-Supervisor

Asst. Prof. İhsan ÇİÇEK

ISTANBUL, 2024

Contents

1 Introduction **1**

1.1 Machine Learning Algorithms 2

1.1.1 Supervised Learning 3

1.1.2 Classification 3

1.1.2.1 Logistic regression 4

1.1.2.2 Support Vector Machine 4

1.1.2.3 Artificial Neural Networks 4

1.1.2.4 Decision Trees 5

1.1.3 Regression 6

1.1.3.1 Linear 6

1.1.3.2 Polynomial 7

1.1.4 Deep Learning Algorithms 7

1.1.4.1 Convolution Neural Network 8

1.1.4.2 Recurrent Neural Networks 9

1.1.5 Unsupervised Learning 10

1.1.5.1 Clustering Algorithm 11

1.1.5.2 K-Means 11

1.1.5.3 Dimensionality Reduction 11

1.1.5.4 Principal Component Analysis 12

1.1.6 Semi-Supervised Learning 13

1.1.6.1 Self Training 13

1.1.6.2 Low-density Separation Models 14

1.1.7 Reinforcement Learning 15

1.1.7.1 Dynamic Programming 15

1.1.7.2 Monte Carlo Methods 15

1.2 Motivation 16

1.3 Literature Review for A/DAS Application 20

1.4 Thesis Statement and Contributions 29
1.5 Thesis Organization 31

2 Technical Background of Neural Network Computation 34
2.1 Matrix-Vector Operations 34
2.2 Optimization Algorithms 34
2.3 Software Environment 35

3 Recent Advances in Embedded Systems 36
3.1 ML Tools for Embedded Platforms 36
3.2 A/DAS Applications 39
3.2.1 Traffic Sign Detection 40
3.2.2 Road & Lane Detection 42
3.2.3 Vehicle & Pedestrian Detection 46
3.2.4 Driver's Behaviours 50
3.2.5 Including More Than one A/DAS Task 54
3.3 A/DAS Application Challenges 58
3.3.1 Power Consumption 59
3.3.2 Execution Time 61
3.3.3 Accuracy 62
3.3.4 Memory 63
3.3.5 Scalability 64
3.3.6 Applicability and Flexibility 65
3.3.7 Bottlenecks and Potential Solutions 65
3.4 Optimization Techniques for Hardware and Software 66
3.4.1 Fixed and Floating Point Arithmetic 66
3.4.2 Platforms 67
3.4.3 Software Optimization Techniques 68
3.4.4 Compression 71

3.4.5 Aim-Oriented Design 72
3.4.6 Tests for Optimal Solution 72

4 Optimizing Performance: Hardware Acceleration Across Diverse Embedded Platforms 74

4.1 PYNQ-Z1 Applications 74
4.1.1 Performance evaluation of real-time video processing edge detection on various platforms 74
4.1.1.1 Hardware and Software Design 77
4.1.1.2 Test and Evaluation 80
4.1.1.3 Results and Discussion 80
4.1.2 Hardware Acceleration of FIR filter Implementation on ZYNQ SoC 81
4.1.2.1 Discrete time FIR filter design 82
4.1.2.2 Hardware and software Design 85
4.1.2.3 Test and Evaluation Results 86
4.1.2.4 Conclusions & Discussion 88
4.1.3 Real-time Hardware Accelerator of Low Precision Quantized Custom NN Model on ZYNQ SoC 89
4.1.3.1 Setup, Training and Implementation of the Model 91
4.1.3.2 Performance Analysis 94
4.1.3.3 Conclusion and Discussion 97

5 AMD Xilinx Vitis-AI 99

5.1 General Framework of A/DAS Multi-Task Application 99
5.1.1 Custom DPU-TRD Design 101
5.1.1.1 Design Phases 102
5.1.1.1.1 Vivado 2022.1 IDE 102
5.1.1.1.2 PetaLinux 2022.1 104
5.1.1.2 Hardware Configuration 111

5.1.1.3 Software Configuration . . . 113
5.1.2 Multi-Learning Model . . . 115
5.1.2.1 Shared Backbone Encoder . . . 117
5.1.2.2 Segmentation Head . . . 118
5.1.2.3 Detection Head . . . 119
5.1.2.4 Depth Estimation Head . . . 120
5.1.3 Quantization Aware Training (QAT) . . . 121
5.1.4 Datasets . . . 126
5.1.5 Model Training . . . 128
5.2 Energy Efficiency Assessment in Advanced Driver Assistance Systems with Real-Time Image Processing on Custom Xilinx DPUs . . . 131
5.2.1 Experimental Results . . . 132
5.2.1.1 Power Consumption & Inference Efficiency . . . 133
5.2.1.2 Resource Utilization . . . 135
5.3 Integration and Optimization of A/DAS Multi-Task Learning Applications on MPSoC-FPGA Platforms through Custom DPU Architectures . . . 137
5.3.1 Design and Implementation of a Real-Time Hardware Accelerator on ZYNQ UltraScale+ MPSoC for A/DAS . . . 138
5.3.1.1 Experimental Setup . . . 138
5.3.1.1.1 System setup . . . 138
5.3.1.1.2 Datasets . . . 139
5.3.1.1.3 Model training . . . 140
5.3.1.2 Results and Discussion . . . 141
5.3.2 Real-Time Multi-Task A/DAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture . . . 147
5.3.2.1 Experimental setup . . . 147
5.3.2.1.1 System setup . . . 147
5.3.2.1.2 Datasets . . . 147

5.3.2.1.3 Model training 148
5.3.2.2 Results and Discussion 149
5.3.3 Real-Time Multi-Learning Deep Neural Network on an MPSoC-FPGA for Intelligent Vehicles Harnessing Hardware Acceleration with Pipeline 153
5.3.3.1 Experimental Setup 153
5.3.3.1.1 System Setup 153
5.3.3.1.2 Dataset 154
5.3.3.1.3 Model Training 155
5.3.3.2 Results and Performance Evaluation 156

6 Conclusion & Discussion **161**

6.1 Detailed Insights from the Studies 161

7 Future Work **164**

ÖZET

GELİŞMİŞ SÜRÜCÜ DESTEK SİSTEMLERİNİN (A/DAS) DONANIM UYGULAMASI İÇİN MAKİNE ÖĞRENİMİ ALGORİTMALARININ TASARIM YÖNLERİ

Bu tez, gelişmiş donanım-yazılım entegrasyonu ve çoklu görev öğrenme çerçeveleri aracılığıyla Gelişmiş Sürücü Destek Sistemlerini (A/DAS) geliştirmeye yönelik yeni bir yaklaşım sunmaktadır. Pynq-Z1 ve esas olarak Kria KV260 Vision geliştirme kartları kullanılarak yürütülen araştırma, Python tabanlı PYNQ paketi ve FINN çerçevesi gibi hesaplama çerçevelerinin Pynq-Z1 kartına entegrasyonunu tanıtmaktadır. Bu yenilikçi yaklaşım, çeşitli uygulamalarda önemli donanım hızlandırmasını gösteren deneyler için bir temel oluşturmakta ve alana benzersiz bir katkı sağlamaktadır.

Tezin merkezinde, tamamen otonom sürüş için A/DAS'de güvenlik ve verimliliği artıran bir MPSoC-FPGA mimarisi üzerinde donanım-yazılım ortak tasarımı yer almaktadır. Kria KV260 üzerinde çeşitli algılama ve segmentasyon görevlerinin ayrıntılı bir şekilde uygulanması yoluyla, çalışma sistem performansında önemli bir iyileşme göstermekte ve kaynakların ve gücün verimli yönetimini tartışmaktadır. Ayrıca tez, heterojen mimariler üzerinde çok görevli bir A/DAS uygulamasını detaylandırarak güç ve kaynak kullanımında önemli azalmalarla kayda değer bir hesaplama verimliliği elde etmektedir.

Genel olarak bu araştırma, sistem doğruluğu ve bellek verimliliği için yeni ölçütler sağlayarak akıllı araç teknolojileri alanını derinden etkilemekte ve A/DAS'de gelecekteki ilerlemeler için zemin hazırlamaktadır. Açıklanan deneyler, önerilen yaklaşımların gerçek dünya senaryolarına uyarlanabilirliğini ve etkinliğini göstermekte ve sonuç olarak A/DAS literatürüne önemli bir katkı sağlamaktadır.

July 26, 2024 **Güner TATAR**

ABSTRACT

DESIGN ASPECTS OF MACHINE LEARNING ALGORITHMS FOR THE HARDWARE IMPLEMENTATION OF ADVANCED DRIVER ASSISTANCE SYSTEMS (A/DAS)

This thesis presents a novel approach to enhancing Advanced Driver Assistance Systems (A/-DAS) through advanced hardware-software integration and multi-task learning frameworks. The research, conducted using the Pynq-Z1 and mainly Kria KV260 Vision development boards, introduces the integration of computational frameworks like the Python-based PYNQ suite and the FINN framework on the Pynq-Z1 board. This innovative approach establishes a foundation for experiments demonstrating significant hardware acceleration in diverse applications, a unique contribution to the field.

Central to the thesis is the hardware-software co-design on an MPSoC-FPGA architecture, which enhances safety and efficiency in A/DAS for fully autonomous driving. Through detailed implementation of various detection and segmentation tasks on the Kria KV260, the study shows a substantial improvement in system performance and discusses the efficient management of resources and power. Additionally, the thesis details a multi-task A/DAS implementation on heterogeneous architectures, achieving notable computational efficiency with significant reductions in power and resource usage.

Overall, this research profoundly impacts the intelligent vehicle technologies domain by providing new benchmarks for system accuracy and memory efficiency, setting the stage for future advancements in A/DAS. The experiments described illustrate the adaptability and effectiveness of the proposed approaches in real-world scenarios, ultimately making a significant contribution to the literature on A/DAS.

July 26, 2024 **Güner TATAR**

CLAIM FOR ORIGINALITY

This thesis represents my original work under my academic supervisors' guidance. The research presented herein is the completion of extensive study and analysis, drawing upon approximately 300 international studies to provide a comprehensive relative framework across diverse Advanced Driver Assistance System (A/DAS) tasks, such as "Traffic Sign Detection," "Road and Lane Detection," "Vehicle and Pedestrian Detection," "Including More Than one A/DAS Tasks," and "Driver Behavior." These tasks were meticulously analyzed and cataloged, offering insights into less explored areas and highlighting roads for future investigations.

Central to this thesis is the development of innovative multi-task learning models by modifying the hyperparameters of the multi-learning model provided by AMD Xilinx and running them on a custom overlay. The construction and implementation of Custom Deep Learning Processing Units (DPUs) facilitated the creation of a cost-effective architecture capable of handling multiple A/DAS tasks simultaneously on a single hardware platform. This involved the parallel development of the processor system (PS), programmable logic (PL), and processing engines (PE), optimizing performance while minimizing power consumption and memory use. The research has significant practical implications, enabling more efficient and cost-effective A/DAS systems.

The research contributions are distinct and significant. They are marked by developing a unique custom overlay for the A/DAS applications with a specifically designed DPU that enhances computational efficiency and hardware utilization. Furthermore, techniques such as 8-bit model quantization were implemented to balance computation efficiency with model accuracy, alongside executing a multi-tasking, adaptable A/DAS application on the Custom DPU.

A novel Hardware-Software co-design methodology was applied to seamlessly integrate hardware and software components, enhancing system performance and resource utilization. Moreover, an original Performance Index (PI) criterion was introduced to enable accurate, comprehensive performance comparisons across studies, addressing a notable

gap in the existing literature. This thesis also extends to detailed frequency analysis, assessing the energy efficiency of all Custom Xilinx DPU-TRDs for A/DAS multi-task learning applications, and conducting performance comparisons on the Kria KV-260 vision board.

In addition, some performance comparisons of special applications were conducted on the PYNQ-Z1 development board using frameworks like FINN and Python productivity PYNQ to guide interest.

The work herein has not been submitted for any other degree or professional qualification and is my original research. It is a real contribution to the field, filling existing gaps and pushing the boundaries of knowledge in intelligent vehicle technology and A/DAS. The impact of this research on the field is substantial, as it significantly advances our understanding of A/DAS and opens up new avenues for exploration.

July 26, 2024 **Güner TATAR**

SYMBOLS

$I(x, y)$	:Pixel density
h	:Distance
$Lf(x)$	:Discrete Laplacian operator
l	:Filter
∇	:Gradient operator
η	:Learning rate
∇_{J_c}	:Jaccard loss extension
C	:Class
$x^{(i)}$	:Training sample
$y^{(i)}$	:Label
t	:Time (s)
f	:Frequency (Hz)
$\sum$	:Summation operator
$\prod$	:Product operator
$x[n]$	:Input signal
$y[n]$	:Output signal
M	:Filter order
ω_p	:Pass-band corner frequency
ω_s	:Stop-band corner frequency
ω_c	:Critical frequency
$h_d[n]$	:Ideal impulse response
$m(c)$	:Vector of errors
e^{α}	:Optimal scaling to align
$\frac{\partial}{\partial x}$	:Partial derivative
f_q	:Quantization function
f_d	:De-quantization function

x_q	:Quantized matrices
y_q	:Floating-point
z_x and z_y	:Zero points
∇_x	:Undetermined small value
π	:Constant (≈ 3.14)
s	:Positive floating-point scale
exp	:Exponential
log	:Logarithm
n	:Count of elements
$\sigma(z)$	:Logistic sigmoid function
β_1	:Exponential decay rate
n_{ii}	:True positive
n_{ji}	:False negative
t_i	:Total count of pixel
n_{cl}	:Total number of classes

ABBREVIATIONS

A/DAS	:Advance Driver Assistance Systems
MPSoC	:Multi Processors System on Chip
FPGA	:Field Programmable Gate Arrays
GPS	:Global Positioning Systems
GPUs	:Graphics Processing Units
CPUs	:Central Processing Units
AXI-DMA	:Advanced eXtensible Interface - Direct Memory Access
ALU	:Arithmetic Logic Unit
BRAM	:Block Random Access Memory
URAM	:Ultra Random Access Memory
OCM	:On-Chip Memory
DSP	:Digital Signal Processing
DDR	:Double Data Rate
FFs	:Flip-Flops
BUFG	:Global Clock Buffer
LUTs	:Look-Up Tables
LUTRAMs	:Look-Up Table Random Access Memory
CLBs	:Configurable Logic Blocks
Vivado-IDE	:Vivado - Integrated Development Environment
Vitis-AI	:Vitis - Artificial Intelligence
HLS	:High-Level Synthesis
Hw	:Hardware
Sw	:Software
OpenCV	:Open Computer Vision Library
PL	:Programmable Logic
PS	:Processing Systems

PE	:Processing Engines
SIMD	:Single Instruction Multiple Data
BAMs	:Bidirectional Associative Memories
CNN	:Convolutional Neural Network
DNN	:Deep Neural Network
RNN	:Recurrent Neural Networks
FNN	:Feed-Forward Neural Networks
MNN	:Modular Neural Network
SNN	:Spiking Neural Network
QNN	:Quantized Neural Network
BNN	:Binary Neural Network
FINN	:Experimental Framework
STE	:Straight-Through Estimation
LSQ	:Learned Step-Size Quantization
AP	:Average Precision
mAP	:mean Average Precision
IoU	:Intersection over Union
mIoU	:mean Intersection over Union
SILog	:Scale Invariant Logarithmic
GOPs	:Giga Operation Per Second
SGD	:Stochastic Gradient Descent
BCELoss	:Binary Cross Entropy Loss
ELMs	:Extreme Learning Machines
DL	:Deep Learning
ML	:Machine Learning
AI	:Artificial Intelligence
SVM	:Support Vector Machine
TSVM	:Transductive Support Vector Machine

RL	:Reinforcement Learning
MDP	:Markov Decision Process
NLP	:Natural Language Processing
DP	:Dynamic Programming
LR	:Logistic Regression
PCA	:Principal Component Analysis
SSL	:Semi-Supervised Learning
SSD	:Single Shot Multi-Box Detector
DWSC	:Depth-Wise Separable Convolution
DPU	:Deep Learning Processing Unit
TRD	:Target Reference Design
IPs	:Intellectual Properties
HPC	:High Performance Computing
QAT	:Quantization Aware Training
PTQ	:Post Training Quantization
HPC	:High-Performance Coherent
ACP	:Accelerator Coherency Port
HP / HPM	:High-Performance Ports
AXI HPM FPD	:AXI High-Performance Master Full Power Domain
AXI HPM LPD	:AXI High-Performance Master Low Power Domain

LIST OF FIGURES

Figure 1.1 Artificial intelligence and subbranches 2
Figure 1.2 Supervised learning 3
Figure 1.3 Classification of two groups with SVM technique 5
Figure 1.4 Overview of artificial neuron or node 6
Figure 1.5 Decision tree classification algorithm 6
Figure 1.6 Simple linear and polynomial regression overview 7
Figure 1.7 a) CNN, and b) RNN structure 8
Figure 1.8 Unsupervised learning in ML 10
Figure 1.9 K-means clustering implementation 12
Figure 1.10 Dimension reduction of principal component analysis 13
Figure 1.11 Semi-supervised learning approach 13
Figure 1.12 A review for tsvm and svm 14
Figure 1.13 Reinforcement learning algorithm in ML 15
Figure 1.14 Basic overview of the ML algorithms [1–4] 18
Figure 4.1 Reference graph of the pixels which goes to zero 76
Figure 4.2 A 5 by 5 kernel for a Laplacian filter 76
Figure 4.3 Simplified flow chart of video streaming steps 77
Figure 4.4 Basic interface view with the pipeline of computer vision overlay . 78
Figure 4.5 Convolution operation with 5x5 kernel 79
Figure 4.6 Post implementation of ZYNQ architecture resource usage in graph 81
Figure 4.7 ARM+FPGA-based accelerator for a Laplacian high-pass filter a) filtered image, b) inverse image 81
Figure 4.8 FIR Filter Block Diagram 83
Figure 4.9 Low pass filter frequency response [5] 84
Figure 4.10 Flowchart of obtaining the signal of interest. 86

Figure 4.11 Operation of the low pass FIR filter; a) Noisy input signal, b) The filtered signal on the bare ARM processor, c) The filtered signal on ARM+FPGA hybrid use 86
Figure 4.12 Hardware accelerator overlay design. 87
Figure 4.13 Low pass FIR filter with 2MHz cutoff frequency. 87
Figure 4.14 A simplified representative QAT 90
Figure 4.15 Algorithmic flowchart of the model 93
Figure 4.16 A simplified representative multi-layer perceptron 95
Figure 4.17 Low precision int-2 quantized custom model cost function and test accuracy at 100 epochs: (a) cost function, (b) training accuracy . . 96
Figure 4.18 An overview of real-time implementation on ZYNQ SoC. 97
Figure 5.1 General overview of the A/DAS multi-task framework 100
Figure 5.2 Vitis heterogeneous embedded system design flow 103
Figure 5.3 Custom DPU-TRD overlay design 104
Figure 5.4 Separable dept-wise convolution architecture. 114
Figure 5.5 An overview of quantization and compilation of fp32 bits NN model 115
Figure 5.6 An overview of multi-task learning network 116
Figure 5.7 SSD single shot multi-box detector [6] 120
Figure 5.8 Forward and backward pass for QAT process 123
Figure 5.9 A part of quantized and compiled multi-learning framework 130
Figure 5.10 Experimental setup 131
Figure 5.11 Performance comparison of proposed study based on frequency and DPU architecture 134
Figure 5.12 Resource utilization and instantaneous power consumption for each DPU architecture 134
Figure 5.13 General overflow of the multi-task learning model on Kria KV260 with K26-SOM 139
Figure 5.14 Kernel pipeline architecture 140

Figure 5.15 Real-time implementation of proposed study 142
Figure 5.16 Cognitive analysis of object detection and segmentation performance across varied weather conditions 144
Figure 5.17 Real-time implementation of A/DAS multi-task learning 150
Figure 5.18 Performance evaluation of DDR ports read/write in Vitis AI analyzer 153
Figure 5.19 Experimental setup of proposed study 154
Figure 5.20 Simultaneous training process for each task 156
Figure 5.21 Implementation results of proposed multi-learning framework in real-time 157
Figure 5.22 Performance analysis of the multi-learning framework under snowy night condition. 158
Figure 5.23 Performance analysis of proposed multi-learning framework under rainy night condition. 159

LIST OF TABLES

Table 1.1 A summary of ML algorithms available for use in A/DAS applications [7–10] 20

Table 1.2 Embedded platforms suitable for A/DAS applications and their features 21

Table 1.3 Comprehensive overview of computational tasks and hardware platforms in A/DAS and related applications 21

Table 1.4 Advancements in A/DAS: A comprehensive exploration of multi-learning frameworks, hardware software co-design, and diverse hardware accelerators for multi-task applications 23

Table 3.1 A List of ml frameworks available for use in specific applications [11–17] 37

Table 3.2 CNN Models & computational challenges for specific applications . 38

Table 3.3 ToolFlow features in specific applications 38

Table 3.4 Performance comparison of A/DAS traffic sign & light detection task in the literature 41

Table 3.5 A/DAS tsd implementation information in the literature 42

Table 3.6 Performance comparison for A/DAS traffic road & lane detection tasks in the literature 46

Table 3.7 A/DAS vehicle & pedestrian detection tasks implementation information in the literature 50

Table 3.8 A/DAS vehicle and pedestrian detection tasks implementation information in the literature 51

Table 3.9 A/DAS driver's behaviour detection tasks and features in the literature 55

Table 3.10 A/DAS driver's behaviour detection task implementation information in the literature 56

Table 3.11 A/DAS multi-tasks and associated features in the literature 59

Table 3.12 Implementation information for A/DAS multi-task applications . . 60

Table 4.1 Comparison of the 5x5 kernel Laplacian high-pass Filter's real-time video processing edge detection performance on various Hardware platforms . . . 80
Table 4.2 Post implementation of ZYNQ architecture resource usage in table 80
Table 4.3 Window functions and their properties [18] . . . 84
Table 4.4 Performance comparison of FIR filter implemented in different design environments . . . 86
Table 4.5 Resource consumption of Zynq Z720 SoC . . . 95
Table 4.6 Details about the computer utilized for calculations . . . 96
Table 4.7 Performance evaluation of real-time implementation on ZYNQ SoC 97
Table 5.1 DPU configuration cases in studies for energy efficiency through frequency analysis . . . 113
Table 5.2 Comparison of resource utilization of the proposed study in the cases 132
Table 5.3 Total resource consumption for each configuration . . . 133
Table 5.4 Use of data-sets . . . 140
Table 5.5 Performance comparison of the studies in terms of specified metrics 143
Table 5.6 Performance comparison of the study in fixed point and floating point arithmetic . . . 145
Table 5.7 Real-time performance comparison of quantized multi-task learning model on different thread size . . . 145
Table 5.8 The resource consumption of hardware architectures employed in similar studies . . . 146
Table 5.9 Use of datasets . . . 148
Table 5.10 Performance comparison of the studies in terms of specified metrics 149
Table 5.11 Performance comparison of the study in int8 and FP32 model types 151
Table 5.12 Performance comparison of DPU with and without optimized . . . 151
Table 5.13 Resource consumption of hardware architectures used in similar studies . . . 152

Table 5.14 DPU configuration for TRD 154
Table 5.15 Use of dataset 155
Table 5.16 Compare the performance of the multi-task studies based on the specified metrics 157
Table 5.17 Exploring the real-time performance differences of quantized multi-learning framework across various thread sizes 159
Table 5.18 Comparative analysis of resource consumption in studies closely related to our multi-learning framework 160
Table 5.19 Performance comparison of the proposed study in int8 and fp32 model types 160

1 Introduction

The advancements in technology are increasingly impacting our daily lives. The digitalization of everything and its connection to internet networks are driving these technological developments. There are seven times more Internet-connected devices than people worldwide, marking a shift towards the Internet of Things (IoT) [19]. Cisco's internet growth and connectivity projections after 2024 suggest continued expansion and evolution in several vital areas [20]. However, the growing density of connected devices will lead to challenges such as data traffic and memory requirements. According to their reports, the number of global internet users, the pervasiveness of 5G technology, and the enhancement of network speeds are expected to rise significantly. Research suggests that the data size in IoT applications will reach the level of zettabytes [21]. Dealing with data of such magnitude will require intelligent machine-to-machine interactions [21, 22].

Moreover, the widespread use of IoT and internet access will have economic implications. According to a report by the McKinsey Global Institute, it is projected that the IoT will contribute around 11.1 trillion US dollars to the global economy by 2025, impacting various sectors such as factories, cities, autonomous vehicles, A/DAS, and even healthcare [23]. By 2025, Cisco envisions a complex internet landscape shaped by various uncertainties, such as investment in broadband infrastructure, technological breakthroughs, and user behavior. These factors could lead to diverse scenarios ranging from pervasive internet growth to challenges like cyber insecurity or economic stagnation affecting technology adoption.

The field of artificial intelligence (AI) focuses on making machines intelligent and enabling them to move independently. AI involves developing machines that can think like humans and imitate their actions. Machine learning (ML) and deep learning (DL) have been the primary focus areas in recent years, representing the core of artificial intelligence. The relationship between artificial intelligence, ML, and DL algorithms is illustrated in Figure 1.1. The American Arthur Samuel first coined the term "machine learning" in 1959. However, the concept did not progress significantly due to limited computer science developments at the time; it only progressed a little. It was not only in the advancement of semiconductor technology and modern processors that computer hardware and software

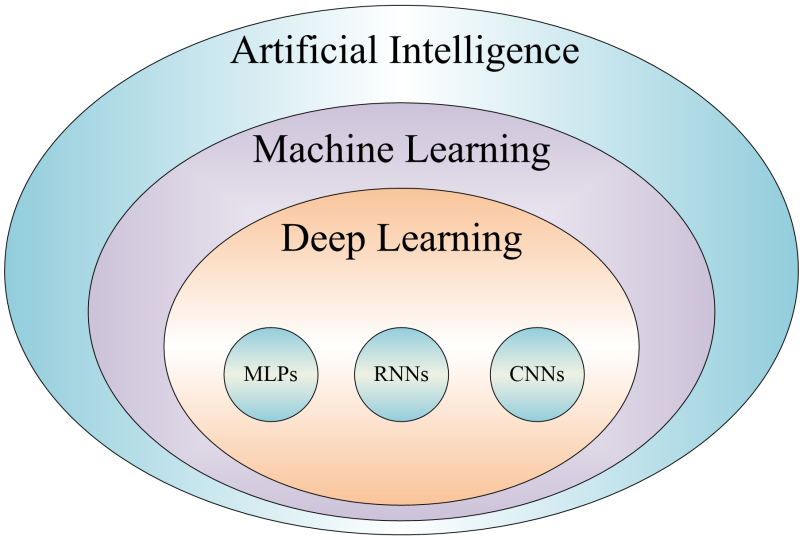


Figure 1.1: Artificial intelligence and subbranches

could support ML algorithms with high computational demands. This progress has led to the emergence of various ML algorithms, frameworks, and hardware acceleration tools designed to enable complex and time-consuming calculations to be performed quickly. These tools use software and hardware accelerators with specific optimizations and algorithms. Hardware accelerators are created by speeding up specific circuit blocks and chip architectures, as well as operational kernels used for specific functions, to balance performance and functional flexibility. To appreciate the benefits of these high-tech products, it is essential to understand ML algorithms and hardware acceleration structures under constraints such as speed, size, power consumption, flexibility, accuracy, and memory in embedded systems and applications. ML accelerators are built with NNs, which require the network to be trained with a large amount of sample data before being used to predict the properties of new data.

1.1 Machine Learning Algorithms

ML is the process that learns with large amounts of data in a process called *training* and then applies less data knowledge to the educated network to analyze and give the result called *inference*. New data is used to improve ML performance, and the algorithm learns and explores the processes of this new data. Over time, the concept of intelligence emerges. Below are briefly mentioned the algorithms used in *training* and *inference* to understand the subject integrity and ML fully.

1.1.1 Supervised Learning

In supervised learning, a model function is deduced from labeled training data, enabling the algorithm to predict new data instances [24–26]. This paradigm of ML, wherein the training set comprises examples, each containing an input object (often represented as a pair (x, y))—with x being the input vector and y the corresponding output value—is utilized to derive a function that predicts output values for future inputs, as depicted in Figure 1.2 [27]. Supervised learning encompasses three primary categories: *classification*, *regression*, and *deep learning*, each of which includes a range of specific algorithms. These categories serve distinct purposes: classification for categorizing data, regression for predicting continuous outcomes, and DL for handling complex patterns in data through layered computations, as exemplified by (1) and (2).

$$y = f(x, label) \tag{1}$$

$$prediction = y(x_1) \tag{2}$$

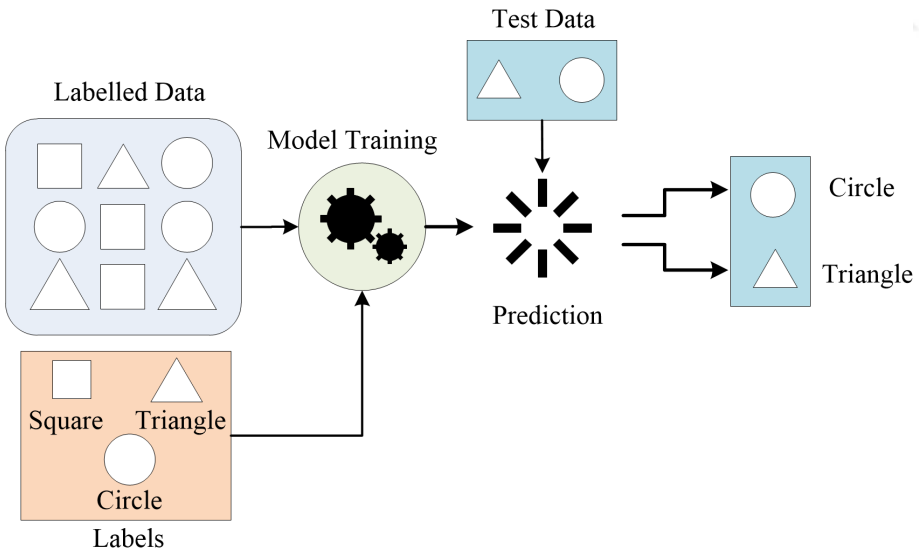


Figure 1.2: Supervised learning

1.1.2 Classification

Classification fundamentally involves recognizing, differentiating, and assigning meaning to objects. Within ML, classification tasks are centered on predicting the class label of

input data samples, a foundational concept in supervised learning [28]. Classification problems can be broadly categorized into four principal types: binary classification, multi-class classification, multi-label classification, and imbalanced classification. Binary classification distinguishes between two distinct classes and is prevalent in widely utilized algorithms such as SVM, logistic regression, and k-NN. This manuscript will clarify the principles and applications of these classification algorithms.

1.1.2.1 Logistic regression LR is commonly applied in ML algorithms, particularly in medical data classification. The model assesses and estimates the weights and coefficients for each data variable, considering a variable vector. Mathematically, the LR function can be defined by the (3), which predicts a multiple linear function [28, 29].

$$y = \frac{e^{b_0+b_1x}}{(1+e^{b_0+b_1x})} \tag{3}$$

Here y is the estimated output, b_0 is the bias value, and b_1 is the coefficient value for the single input value of x.

1.1.2.2 Support Vector Machine SVM is an ML technique commonly used in classification and other learning algorithms. SVM is distinguished by its use of an optimal hyperplane for classification. The resulting hyperplane separates new data points, and the data points that influence the position of the hyperplane are called support vectors [29]. Research [27] has shown that SVM is a more robust classifier than the LR algorithm. The SVM algorithm is a discrimination function that outputs either 1 or 0. Figure 1.3 illustrates significant decision boundaries (hyperplanes) to help understand the SVM technique.

1.1.2.3 Artificial Neural Networks ML has emerged as a pivotal area of research, primarily driven by advances in ANNs and DL technologies. These ANNs are designed to mimic biological NNs' structural and functional aspects, enabling them to learn from complex and uncertain datasets [30–32]. The extensive body of literature on ANNs, as reviewed in [30], primarily explores various frameworks that facilitate the execution of ML algorithms. Among these, ANNs are particularly significant due to their ability to emulate aspects of human cognitive processes. They are adept at tackling diverse chal-

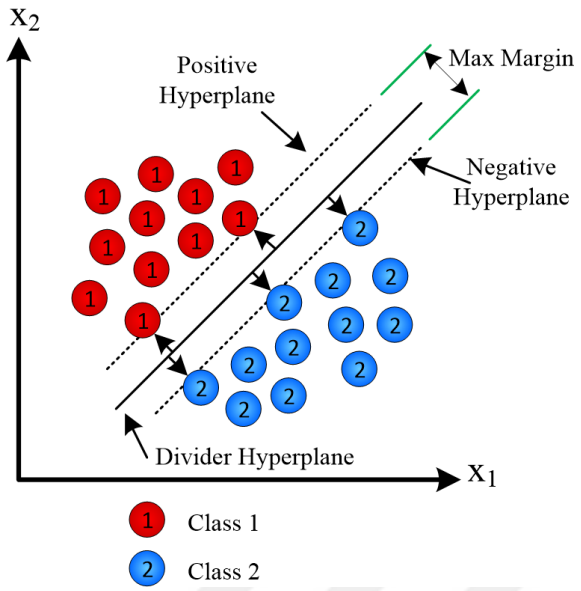


Figure 1.3: Classification of two groups with SVM technique

lenges, including inter-cell association, computation, prediction, inference, and spectrum management.

Furthermore, integrating ANNs into mobile applications and smart devices has significantly enhanced human-machine interactions, positioning these networks as experts in analyzing human-generated data. The classification of ANNs based on their application includes several types: DNNs, which possess multiple hidden layers; Feed-Forward Neural Networks (FNNs), characterized by connections that progress from one neuron to the next without loops; Spiking Neural Networks (SNNs), which closely replicate biological neural activities; Recurrent Neural Networks (RNNs), which permit connections across previous layers to any subsequent layer; Modular Neural Networks (MNNs), which consist of several independent networks linked by an intermediary; and CNNs, which are predominantly employed for visual image analysis. Figure 1.4 illustrates the structure of an artificial neuron or node.

1.1.2.4 Decision Trees A decision tree is a tree-based classification method that is easy to apply and interpret and used to develop predictive models and make decisions using independent variables [33]. While classical decision tree algorithms have fast learning ability and classification speed, they require improvements to be more practical. Figure 1.5 provides a good visualization of how a decision tree algorithm works. Since this thesis

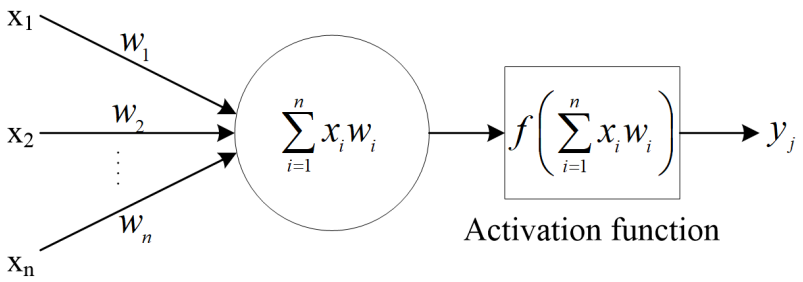


Figure 1.4: Overview of artificial neuron or node

does not focus on decision tree algorithms, I have provided only a superficial overview.

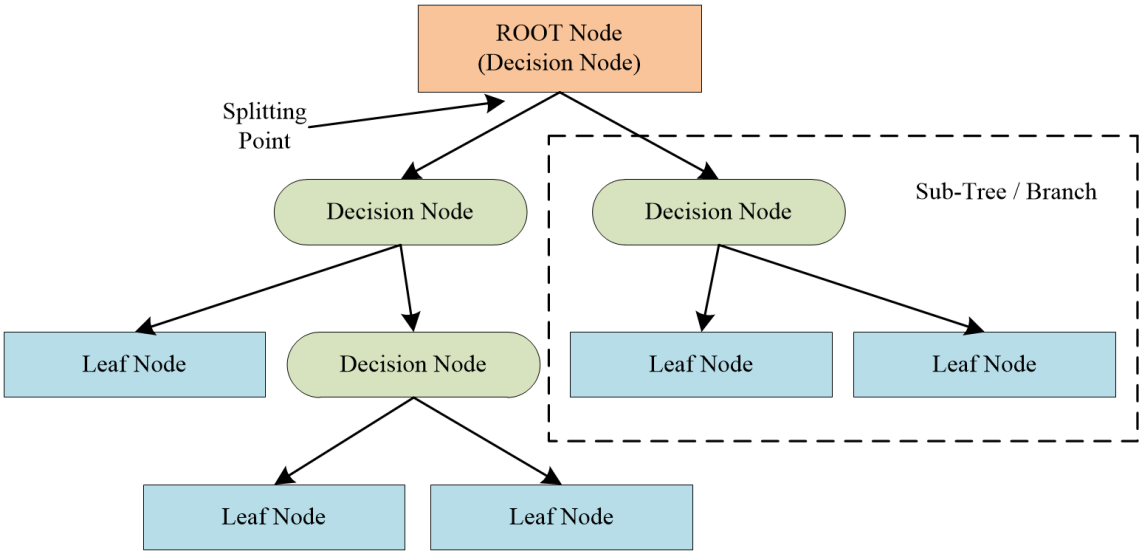


Figure 1.5: Decision tree classification algorithm

1.1.3 Regression

The regression algorithm is an ML algorithm that falls under supervised learning. It works by predicting output based on the characteristics of a given dataset provided to the system's input. In a nutshell, regression is a data mining method that predicts a numerical value [34]. Regression algorithms are widely used in various domains, including financial forecasting, time series forecasting, and trend analysis. Figure 1.6 briefly explains linear and polynomial regression algorithms.

1.1.3.1 Linear Linear regression is commonly known as a method for modeling the relationship between dependent and independent variables [35]. Linear regression is examined in two forms: simple linear regression, which involves a single scalar variable x

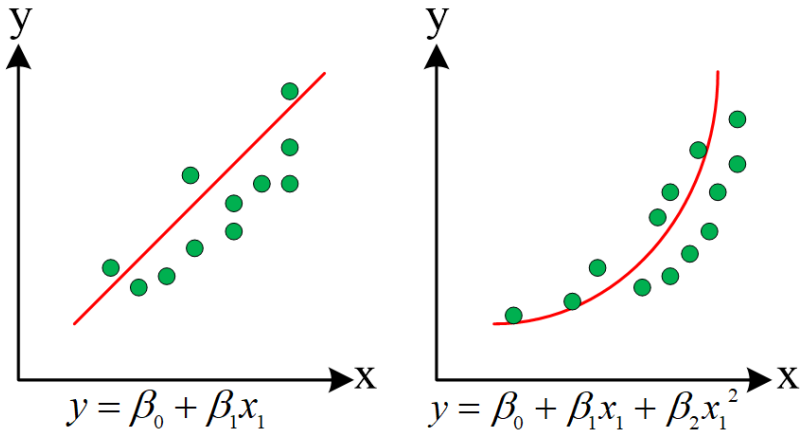


Figure 1.6: Simple linear and polynomial regression overview

and a single scalar response variable y, and multiple linear regression, which affects more than one independent variable.

1.1.3.2 Polynomial Polynomial regression is a type of regression analysis where the relationship between the independent variable x and the dependent variable y is represented as an nth-degree polynomial in x. Unlike linear regression, polynomial regression exhibits a nonlinear relationship between the value of x and the corresponding predicted mean of y.

1.1.4 Deep Learning Algorithms

DL is an essential part of ML algorithms, consisting of multiple processing layers and computational models that aim to mimic the human brain's perception and analysis of multi-modal information [36–38]. DL includes NNs, supervised and unsupervised learning algorithms, and probability models. The research on complex cells, also known as neurons, began in 1943 and led to McCulloch and Pitts' development of ANNs [38, 39]. RNN proposed by Jordan in 1986 and CNNs introduced by LiCun in 1990 marked the era of DL [36, 40].

One of the most impactful breakthroughs in DL occurred in 2006 when Hinton et al., [41] unveiled the Deep Belief Network. This network, composed of multiple Restricted Boltzmann Machines layers, was trained unsupervised [36, 41]. The adoption of DL algorithms has surged, thanks to the use of FPGA or GPU-based high-speed platforms with parallel

processing capability, as opposed to CPU-based serial processing, and publicly labeled datasets. DL has driven computer vision fields such as pattern recognition, object detection, self-driving assistants, visual recognition, NLP, motion tracking, and semantic segmentation to new heights [42–47]. We can briefly explain the most essential DL algorithms: CNNs and RNNs. Figure 1.7 illustrates the structure of CNN (a) and RNN (b) for a better understanding.

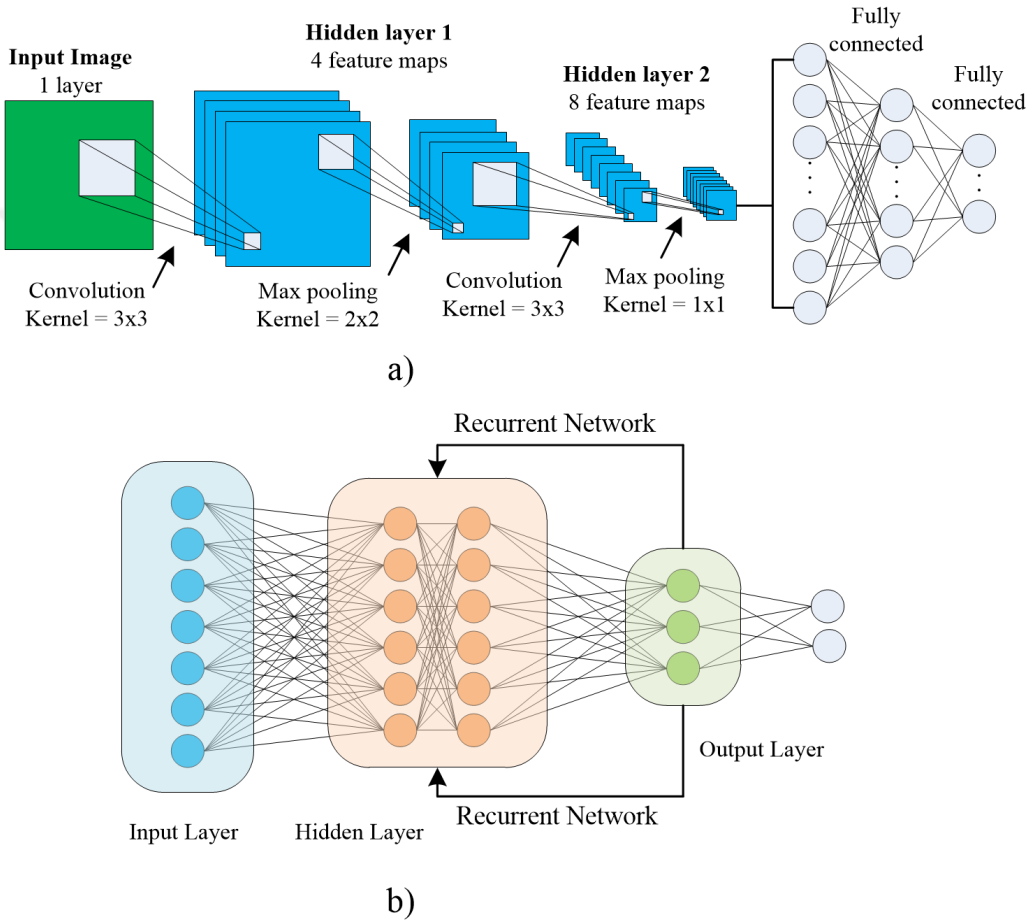


Figure 1.7: a) CNN, and b) RNN structure

1.1.4.1 Convolution Neural Network DL has garnered significant attention in recent developments within AI as a robust methodology capable of managing vast datasets. This popularity surge correlates with performance improvements attributed to increasing the depth of NNs and intense NNs. Among various NN architectures, CNNs have emerged as the outstanding choice for enhancing performance in multiple domains [48–50]. Specifically, [48] offers an expansive view, asserting that CNNs demonstrate superior efficacy

in ML challenges. Furthermore, [48] elaborates on using CNNs in processing image data, highlighting their pivotal role in significant image classification datasets, computer vision, and NLP, where the results have been remarkable. Structurally, a CNN typically comprises three key components: a convolutional layer, a pooling layer, and a fully connected layer. The convolutional and pooling layers are instrumental in feature extraction from the input data, whereas the fully connected layer is tasked with classification. The convolutional layer, characterized by its parameters, performs mathematical operations such as *convolution* (4) and (5), a linear operation essential for data processing [51].

$$\text{Convolution Operation: } O = I * K \tag{4}$$

$$O[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} I[u,v]K[i-u, j-v] \tag{5}$$

I is a 2-D image, K is a 2-D kernel,, and O is an an output feature map.

In the context of CNN applications, digital image pixels are organized into a two-dimensional grid structure. A distinctive feature of CNNs is extracting image features by applying smaller grid segments, known as kernels. These kernels systematically traverse the grid to construct features from each grid unit, as illustrated in Figure 1.7-a. This methodological approach renders CNNs exceptionally effective in image processing applications, consistently delivering robust outcomes. Training of CNN models employs optimization algorithms such as backpropagation and gradient descent, pivotal in refining the model's accuracy and efficiency [48, 52, 53].

1.1.4.2 Recurrent Neural Networks RNNs have been a significant area of research since the 1990s [54]. RNNs are specialized for learning from sequences [55–57] and are adept at handling sequential, non-sequential, or time-varying patterns. Examples of such architectures include Hopfield networks and Bidirectional Associative Memories (BAMs) [54, 58]. RNNs are characterized by their supervised learning framework, consisting of artificial neurons interconnected through feedback loops, allowing for the persistence of information. This architecture includes nonlinear, multi-dimensional hidden layers, often called the network's memory. The state of these hidden layers depends on their preceding states, thereby enabling the retention and utilization of complex infor-

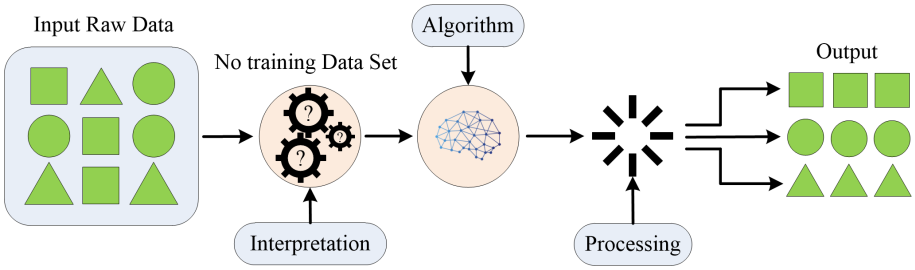


Figure 1.8: Unsupervised learning in ML

mation over extended periods [59].

Similar to CNNs, significant advancements in RNN training have been achieved by developing optimization techniques such as backpropagation and gradient descent. Practical training of RNNs, particularly those incorporating feedback loops as depicted in Figure 1.7-b, requires appropriately structured datasets. These datasets are essential for minimizing the discrepancy between the desired and actual output by optimizing the network's weights during training.

RNNs, like other NNs, employ activation functions to determine their output. These functions are crucial mathematical operations that normalize the network output within specific ranges, such as [1, -1] or [0, 1]. While linear activation functions like the *binary step* and *linear activation* are utilized, nonlinear functions such as *sigmoid* (σ), *tangent hyperbolic* ($tanh$), and the commonly used *rectified linear unit (ReLU)* are preferred for processing complex data, enabling the network to derive meaningful outputs and function effectively.

1.1.5 Unsupervised Learning

Contrary to supervised learning, unsupervised learning, often called 'learning without a teacher,' leverages unlabeled data to discern inherent patterns within the dataset. This learning paradigm identifies patterns based on their probability density or neuronal predilections, as depicted in Figure 1.8. Unsupervised learning encompasses three primary methodologies: clustering algorithms, dimensionality reduction, and anomaly detection. Each method is crucial in exploring data structure without pre-defined labels, facilitating a deeper understanding of the dataset's intrinsic characteristics.

1.1.5.1 Clustering Algorithm Clustering algorithms represent a fundamental class of unsupervised ML techniques extensively utilized for statistical data analysis and data science applications. When applied, clustering algorithms enable extracting valuable insights from datasets by categorizing individual data points into distinct groups based on their similarities. These algorithms function by assessing the similarity across all pairs of samples within the dataset. Consequently, it is crucial to ascertain whether the selected clustering algorithm is appropriately scaled to the dimensions of the dataset. Among the myriad clustering techniques available, the *K-means* clustering algorithm is particularly noteworthy and is extensively discussed in this paper due to its widespread adoption and effectiveness.

1.1.5.2 K-Means The k-means clustering algorithm, which has been used since the 1950s, is arguably the most renowned unsupervised ML algorithm [60–63]. This algorithm partitions a dataset into K clusters by minimizing the sum of the distances between each data point and its nearest centroid, as illustrated in Figure 1.9, which provides a comparative view of the dataset 'before and after' the application of k-means clustering. The literature on k-means clustering is extensive, highlighting its rapid computational speed and simplicity, which facilitate the processing of large datasets [60, 64].
However, k-means has its disadvantages. One significant area for improvement is the necessity to predetermine the number of clusters, which can restrict the algorithm's flexibility and adaptability in exploratory data analysis. Additionally, the initial selection of cluster centers is random, leading to potential inconsistencies across different implementations. These drawbacks may necessitate the consideration of alternative clustering algorithms that are better suited to specific applications, particularly in scenarios where obtaining insightful and reliable clustering is paramount.

1.1.5.3 Dimensionality Reduction As the term implies, dimensionality reduction—or simply dimension reduction—facilitates the transformation of data from a high-dimensional space to a lower-dimensional space [65, 66]. Managing data in high-dimensional spaces is often computationally challenging and analytically complex. Dimension reduction techniques, which can be linear or nonlinear, are typically employed as an intermediate step in

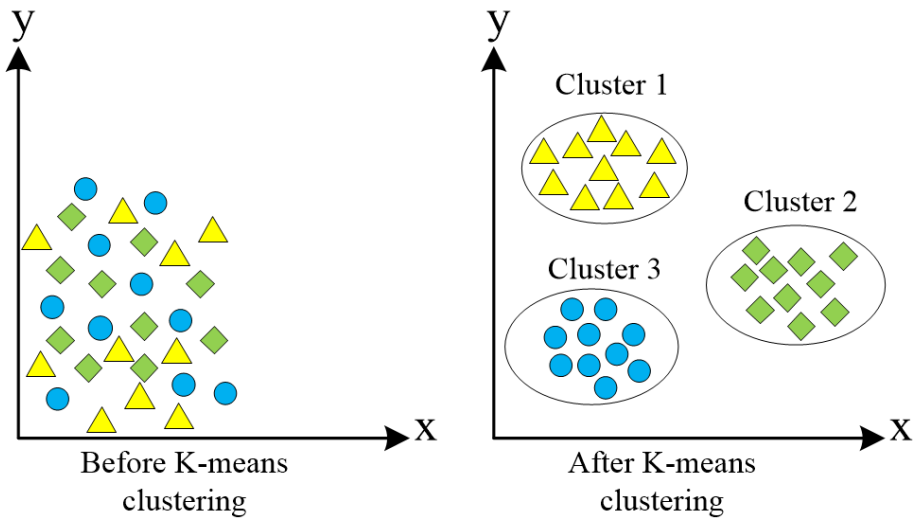


Figure 1.9: K-means clustering implementation

various analytical processes, including cluster analysis, noise reduction, and data visualization. These techniques find extensive applications across multiple fields, such as image processing, signal processing, and speech recognition.

This study provides a concise overview of Principal Component Analysis (PCA), the primary linear method for dimensionality reduction [67]. This discussion is intended to equip the reader with a fundamental understanding of PCA and its applicability in reducing the complexity of high-dimensional data.

1.1.5.4 Principal Component Analysis PCA is widely recognized as the most popular linear and statistical technique for data analysis, deriving groups of independent variables from dependent ones and effectively reducing data dimensions [68–70]. As depicted in Figure 1.10, the PCA process is highlighted, illustrating its operational steps. The scholarly contributions on PCA, extensively discussed in [68], focus primarily on its utility for dimensionality reduction. To reduce the dimensions of an n-dimensional dataset with inputs $x^{(1)}, x^{(2)}, x^{(3)}, \ldots, x^{(m)}$ to a smaller size k, where k is significantly less than n, employing PCA involves the following procedural steps:

Step 1 Gathering the data from *mxn* matrix.

Step 2 Calculate the covariance matrix.

Step 3 Calculate the eigenvector and eigenvalue of Covariance matrix.

Step 4 Picking head parts and shaping a feature vector.

Step 5 Determining the new data set and framing the groups.

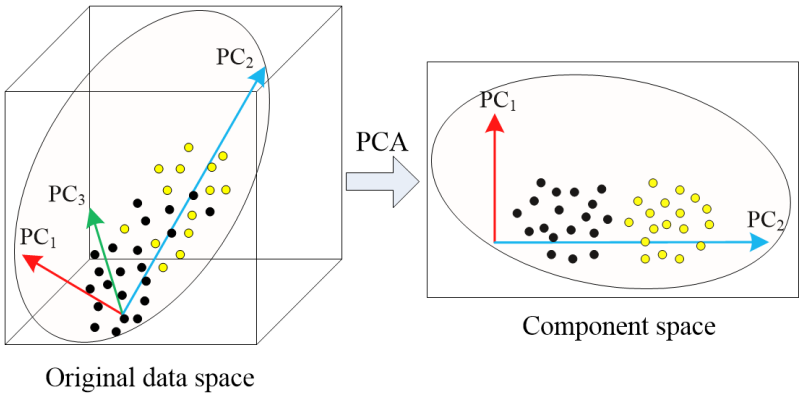


Figure 1.10: Dimension reduction of principal component analysis

1.1.6 Semi-Supervised Learning

SSL algorithm is a methodological approach that mirrors the learning processes observed in natural and computer systems, utilizing both labeled and unlabeled datasets like human learning. Innovations in algorithm design often derive from analyses of how learning behaviors manifest under the dual influence of labeled and unlabeled data, as conceptualized in Figure 1.11. Our understanding of SSL is primarily rooted in empirical research exploring these algorithms' operational dynamics. A notable study by [71] highlights that SSL can enhance learning efficiencies by leveraging a significant volume of unlabeled data alongside a comparatively smaller set of labeled data. The literature on SSL includes various methodologies, among which self-training, also known as self-learning, and the low-density separation model are prevalent.

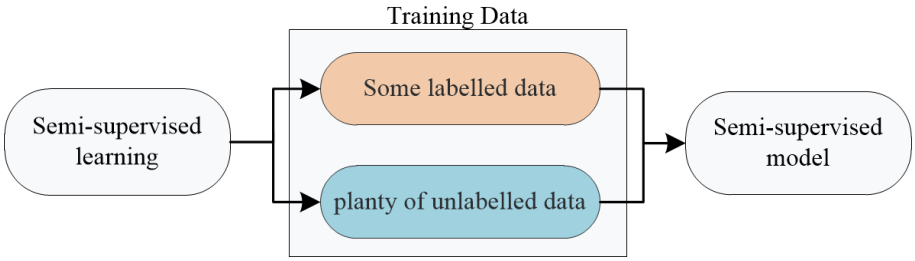


Figure 1.11: Semi-supervised learning approach

1.1.6.1 Self Training Self-learning, a prominent SSL algorithm, has gained acclaim for its effectiveness in educational settings and classification tasks [72–74]. This approach

involves initially training a classifier with a small set of labeled data, expecting the classifier to make predictions after this phase. These initial predictions and newly labeled data are then used to expand the training set recursively. The success of the self-training algorithm hinges on carefully selecting newly labeled data each time the training set is updated [72, 75]. While this method has proven effective in classification, its performance can be compromised by including noisy samples in the training set, leading to erroneous predictions [72].

1.1.6.2 Low-density Separation Models A potential rationale for the effectiveness of semi-supervised learning methods such as the low-density separation model is their ability to outline a decision boundary within a dataset, optimally distinguishing the labeled data class from others. A prominent instance of this model is the TSVM, which operates as a semi-supervised maximal margin classifier. While TSVM shares foundational principles with standard SVM, it performs better in data separation tasks. This distinction is depicted in Figure 1.12, illustrating the enhanced capability of TSVM over traditional SVM in effectively partitioning data. As depicted in Figure 1.12, the green lines represent the

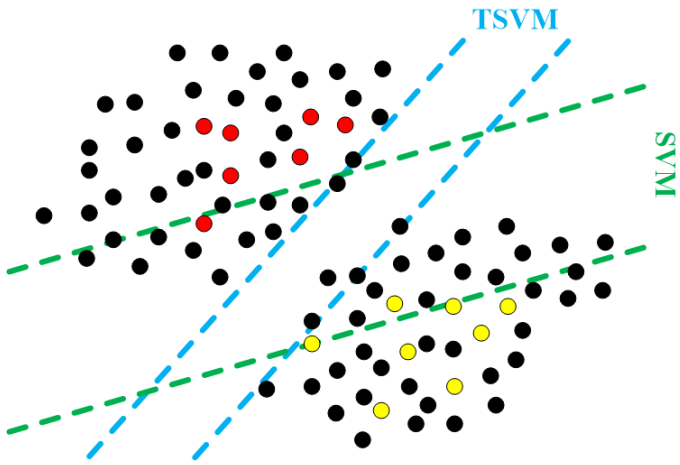


Figure 1.12: A review for tsvm and svm

maximum margin separation of the standard SVM for labeled data. Conversely, the blue lines illustrate the margin separation the TSVM achieves. Notably, the figure demonstrates how TSVM imposes penalties on unlabeled data within the margins, a distinctive feature of its classification strategy. While the given structure effectively manages data in two dimensions, as evidenced by [76], it is essential to recognize that such performance may

not uniformly extend to more complex datasets.

1.1.7 Reinforcement Learning

In broad terms, RL is mathematically modeled using the Markov Decision Process (MDP) [77–79]. This learning paradigm can be conceptualized as a process of developing decision-making capabilities through trial and error. An alternative perspective posits that the effectiveness of RL is augmented by leveraging gains from the hierarchical structures prevalent in real-world environments, thereby facilitating task-specific learning [80, 81]. Figure 1.13 illustrates the operational mechanism of the Reinforcement Learning algorithm, highlighting its dynamic and iterative nature.

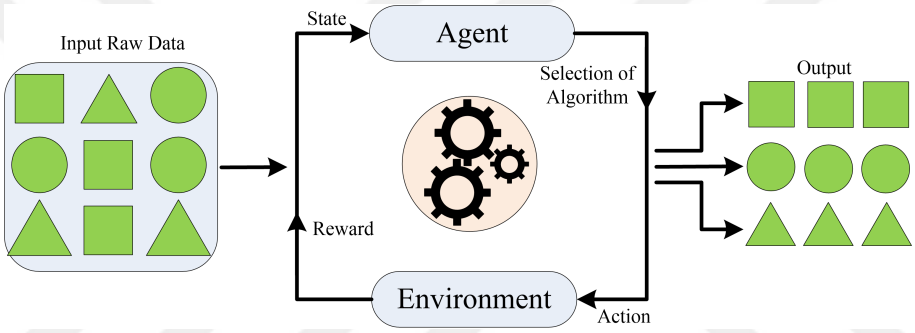


Figure 1.13: Reinforcement learning algorithm in ML

1.1.7.1 Dynamic Programming Introduced by Bellman, DP represents a critical technological tool extensively employed to address the challenges associated with self-learning algorithms [82, 83]. DP is prevalent in various fields, including automatic control systems and operations research, and has gained significant traction in artificial intelligence. Specifically, it is applied in contexts such as inferring from uncertain environments and in the learning and prediction of information. Most strategies developed to mitigate issues in RL algorithms draw upon DP methodologies. When the input data model is ideally configured as an MDP, DP facilitates the derivation of solutions, particularly in scenarios involving a continuous array of action spaces and model states.

1.1.7.2 Monte Carlo Methods The Monte Carlo method, introduced by Ulam and von Neumann in 1947, represents another significant approach to reinforcement learning algorithms. Initially employed for solving integral and differential equations, this method

operates without knowledge of MDP transitions. Learning in the Monte Carlo method is derived from experience [84], with returns or rewards manifesting as random components within the process. The procedural steps employed to ascertain a value function and formulate an optimal policy for a given random policy bear similarities to those found in dynamic programming [85].

Until this point in the thesis, a concise overview of ML algorithms have been presented. The subsequent sections will delineate ML algorithms' methodologies and application domains for A/DAS.

1.2 Motivation

A/DAS play a crucial role in preventing deaths and property damage, evolving from basic functionalities such as parking sensors to complex systems incorporating lane departure warnings, adaptive cruise control, and collision avoidance mechanisms [86]. The evolution of vehicle technology and A/DAS, transitioning from simple, standalone systems to multifaceted ones capable of diverse driving operations, has necessitated integrating various sensors, predictive algorithms, control mechanisms, and collaborative functionalities within and beyond the vehicle [87, 88]. This escalating complexity presents significant challenges in identifying potential failures, diagnosing hardware or software issues, and, critically, ensuring the security and safety of these systems.

Two essential standards, the ISO 6262 Ground Vehicle Functional Safety Standard and the SOTIF ISO/PAS 21448, play a significant role in addressing the challenges of A/DAS. The former focuses on software and hardware failures, while the latter concentrates on the performance limitations of A/DAS under normal conditions [89, 90]. Attaching to ISO 26262 and its ASIL classifications underscores the importance of upholding exacting safety requirements across automotive components. This regulatory framework highlights the delicate balance between operational security and providing advanced functionalities, providing a comprehensive understanding of the challenges and requirements in the field [91].

The emergence of DL [92, 93] and vision-based [94, 95] technology has not just impacted but revolutionized the landscape of autonomous driving [92, 93, 96]. In this era, autonomous driving systems have evolved into complex frameworks, incorporating mul-

tiple sensors and modules tailored for specific functionalities. A fundamental aspect of a robust autonomous driving system lies in its capacity to perceive and react to diverse environmental factors, such as nearby vehicles, pedestrians, and traffic signals. DL has played a transformative role, pushing these capabilities beyond the limits of conventional algorithms. By leveraging DL methods across a spectrum of machine vision tasks, autonomous driving systems have witnessed accuracy, speed, and dependability enhancements that were once unimaginable. The integration of DL into this domain has opened up a realm of possibilities, sparking a new era in the future of autonomous vehicles. While DL finds utility across various domains like healthcare, finance, and entertainment, its prominence is particularly pronounced in A/DAS tasks.

Architectures such as [97], GoogleNet [98], DenseNet, and ResNet [99] have been proposed for image classification, successfully addressing various tasks. In object recognition, multi-stage detectors such as R-CNN [100] and Faster R-CNN [101], alongside single-stage object detection approaches like SSD [6] and YOLO [102], have found practical applications in contemporary scenarios. Similarly, models like [103] and SegNet [104] are lauded for their proficiency in semantic segmentation, while LaneNet [105] and VPGNet [106] have demonstrated success in lane detection tasks.

Figure 1.14 provides an overview of the ML algorithms briefly outlined in Section 1. These algorithms have been applied in diverse research endeavors, each deployed on distinct hardware platforms suitable to specific functionalities, including A/DAS. Incorporating these algorithms into A/DAS tasks represents a significant technological progression within the automotive sector. Furthermore, implementing DL models, such as ANN and CNN, necessitates the judicious selection of hardware accelerators to ensure superior performance and efficient resource allocation [107, 108]. These hardware platforms' careful choice and optimization are vital for maintaining an optimal balance among processing speed, power consumption, precision, and memory demands, underscoring the critical need for a holistic understanding of ML algorithms and hardware architectures. This integration augments the capabilities of A/DAS and raises intricate security and safety challenges that must be addressed to mitigate potential vulnerabilities and guarantee the dependability of these life-critical systems.

A detailed examination of each model reveals that their intricacies can be attributed to

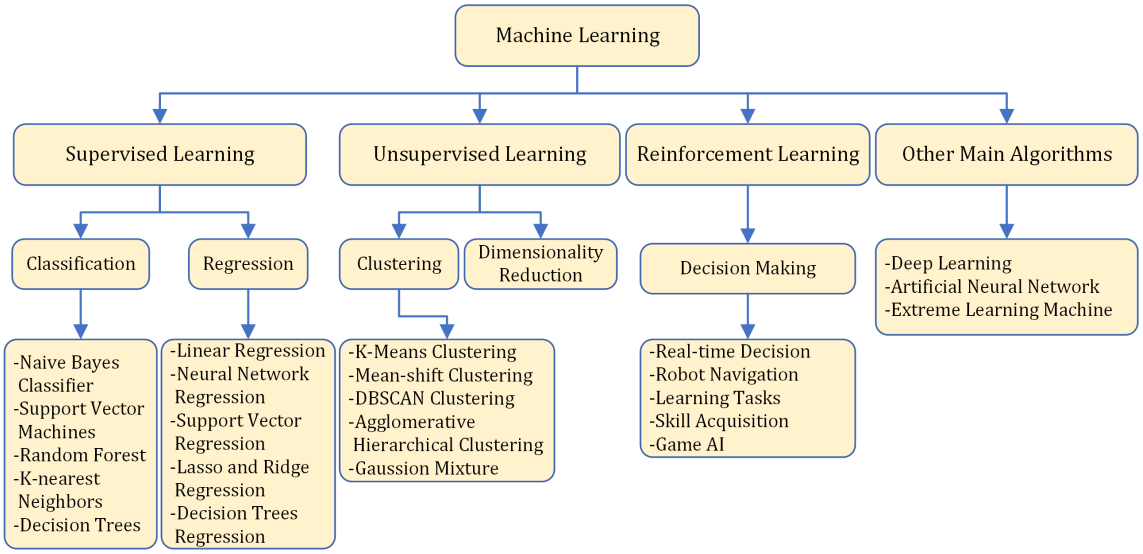


Figure 1.14: Basic overview of the ML algorithms [1–4]

the abundance of layers and parameters. Designers aspire to enhance DNN performance through more extensive architectures, resulting in heightened complexity. This complexity challenges training, extracting insights, and implementing deeper architectures, particularly as requirements escalate. Moreover, employing each specified DL model for a specific task makes their simultaneous use in multi-tasking applications more feasible, enhancing their efficiency across different tasks.

Two distinct strategies emerge as potential solutions to address the complexity of DL models. The first strategy involves running each task-specific model on separate hardware platforms, a solution that comes with inevitable financial implications. On the contrary, the second strategy focuses on enabling and expanding multi-task learning models.

Multi-task learning operates on the assumption that executing numerous interconnected tasks on a single hardware platform using the same infrastructure can boost efficiency. This approach, while it may marginally increase the model's complexity for designers, concurrently reduces the financial burden of running distinct learning models on separate platforms. Therefore, multi-task learning executed through multi-tasking on constrained hardware is a practical and promising strategy, providing hope for a more cost-effective future in A/DAS. This subfield of AI involves executing multiple learning tasks with a single model, leveraging their differences and similarities. This innovative paradigm holds significant promise and potential across various domains. The multi-task learning generally

involves main infrastructures (called backbone) and subheads. Share backbone encoder, which is commonly used can substantially reduce overall computational complexity.

This section delves into a wide range of hardware technologies, including multi-processor architectures, ASICs, GPUs, and FPGAs. These technologies are instrumental in meeting the escalating demands of ML applications, which are increasingly hosted on cloud-based platforms provided by leading technology companies [109]. Despite their extensive capabilities, the CPUs in these cloud environments sometimes need help to efficiently process the vast amount of data required by DL networks. This challenge underscores the practical need to integrate specialized hardware accelerators, such as FPGAs, GPUs, or ASICs, to complement CPUs, thereby significantly enhancing the performance and efficiency of DL tasks [109, 110].

Discussions aim to clarify the diverse hardware configurations utilized across the industry, ranging from multi-core GPUs in general-purpose computing to Google's Tensor Processing Units and Microsoft's FPGA-based systems. The recent shift towards application-specific platforms like FPGAs and ASICs highlights their growing significance, attributed to their parallel processing capabilities, which substantially improve the execution time and accuracy of ML algorithm training.

Rapid advancements in integrated circuit technology tailored to ML applications provide significant performance enhancements. It's important to note that each hardware platform is optimized for different applications and data types, reflecting a nuanced approach to technology selection. As demonstrated in Table 1.1, this thesis delineates the general applications of ML algorithms, underscoring that not every algorithm is suitable for A/DAS applications. ML algorithms are pivotal in these systems due to their high precision and reliability, enabling tasks like adaptive cruise control, lane departure warnings, intelligent speed adaptation, object detection, lane detection, and semantic segmentation. This selective applicability further emphasizes the importance of this research in understanding the integration of ML algorithms within A/DAS contexts.

Recognizing the critical performance expectations for hardware in A/DAS applications, namely speed, power consumption, and latency, we emphasize the role of application-specific hardware accelerators. These accelerators are meticulously evaluated and chosen based on their performance, computational capabilities, energy efficiency, and cost-

effectiveness. Furthermore, leveraging the inherent and interrelated relationships among

Table 1.1: A summary of ML algorithms available for use in A/DAS applications [7–10]

ML Algorithms	Data Type		Purpose and Place of Use		A/DAS Tasks
*	Labelled Data	Unlabelled Data	Classify and Predict	Pattern Recog. and Clustering	*
Logistic Regression	✓	N/A	✓	N/A	N/A
SVM	✓	✓	✓	N/A	✓
ANN	✓	✓	✓	N/A	✓
Decision Trees	✓	N/A	✓	✓	✓
Linear Regression	✓	N/A	✓	N/A	N/A
Polynomial Regression	✓	N/A	✓	N/A	N/A
CNN	✓	✓	✓	N/A	✓
RNN	✓	✓	✓	N/A	✓
K-Means	N/A	✓	N/A	✓	✓
PCA	N/A	✓	N/A	✓	✓
Self-training	✓	✓	N/A	N/A	✓
Low-density Separation Model	✓	✓	✓	✓	N/A
Dynamic Programming	N/A	N/A	N/A	N/A	✓
Monte Carlo Methods	N/A	N/A	N/A	N/A	N/A

various tasks, multi-task learning enhances learning efficiency and predictive accuracy. Optimizing both software and hardware architectures becomes not just important but imperative to enhance processing speed and minimize inference time. Thus, Hardware-Software co-design is critical in supporting and revolutionizing A/DAS within a high performance embedded platforms. Since MPSoC-FPGA presents high performance computing especially in A/DAS applications, this architectures have been preferred in this domain. This approach takes into account both software and hardware, maximizes embedded resources, and achieves high energy efficiency. By cohesively integrating hardware and software design [111–113], A/DAS systems can unlock their full potential, boosting confidence in their efficiency and energy-saving capabilities across various real-world applications.

1.3 Literature Review for A/DAS Application

Recent research has increasingly focused on addressing the specific needs of the automotive industry, including areas like in-vehicle and inter-vehicle communication, autonomous vehicles, traffic and environmental management, and object detection and classification. A significant body of literature explores A/DAS systems, which often employ AI, ML, and DL algorithms on hardware accelerators such as FPGAs, GPUs, ASICs, and multi-core CPUs. The utilization of these hardware components in diverse applications is detailed in Table 1.2 and Table 1.3.

Table 1.2: Embedded platforms suitable for A/DAS applications and their features

Specification	FPGAs	GPUs	CPUs	ASICs
Overview	Re-configurable Computing Flexible IP Blocks	Used in applications with high computational density	Sequential processor for general purpose application	Application specific integrated circuit
Programming	HDL (Verilog-VHDL) C/C++, OpenCL (new) Vitis (newer)	OpenCL & NVIDIA CUDA API allow general purpose programming	Range of high level languages, Assembly language	Application specific; TensorFlow open-source framework for Google's TPU, CPU manufacturers include tools with new ASIC releases
Processor	SoCs include soft or hard IPS Blocks (ARM)	Thousands of identical processor cores	Single and Multi Core MCUs and MPUs	Application-specific third-party IP cores
Peripherals	Configurable I/O Blocks SoCs transceiver blocks	Very limited	Wide range of analogue and Digital peripherals, Digital bus interface	Application-specific industry-standard functions (Ethernet, USBs)
Strengths	Re-configurable High Performans per watt Parallel Processing Wide choice of features: DSPs, CPUs	Very high processing power (Signal, Image and Video processing)	Ease of programming Multitasking and Versatility	Custom-designed for the performance and power consumption balance
Weaknesses	Difficult to program (once) Long development time Poor for floating point operation	High power consumption Necessity of reformulating problems to take advantage of parallelism	Sequential processing, Limited parallelism, OS capability adds high overhead	High cost, Longest development time not reconfigurable

Table 1.3: Comprehensive overview of computational tasks and hardware platforms in A/DAS and related applications

Applications	FPGAs	GPUs	CPUs	ASICs	Keywords	in A/DAS
Computer Vision and Image Processing	✓	✓	N/A	✓	Widely preferred because of the computational intensive use	FPGAs & GPUs
AI training	✓	✓	N/A	N/A	Well-suited for processing big data due to its parallelism	FPGAs & GPUs
AI Inference	✓	✓	✓	✓	FPGAs are generally preferred	FPGAs
Embedded control	✓	N/A	✓	✓	CPUs are widely used because of limited space, low-cost, low power consumption	FPGAs, CPUs, Rarely ASICs
Supercomputer (HPC)	✓	✓	✓	N/A	n supercomputers, it is generally preferred as a CPU, GPU, or a combination of both	N/A
General Purpose Computing	N/A	N/A	✓	N/A	Flexible usage capabilities of CPUs	CPUs
High-Speed Search	N/A	✓	✓	N/A	Google uses TPU ASICs, For coordination & control CPUs, Microsoft's Bing uses FPGAs, For graphics GPUs	CPUs
Industrial Motor Control	✓	N/A	✓	✓	Almost all motor controls have MCUs or ASICs, Recently, FPGAs are among the alternatives	ASICs & FPGAs
Prototyping -Low volume	✓	N/A	N/A	N/A	The best solution thanks to its reconfigurable capability	FPGAs

A/DAS systems integrate software and hardware comprising sensors, transducers, actuators, and processors for high-frequency multiprocessing to prevent accidents in driver-driven and autonomous vehicles. Engineers employ advanced camera systems, LIDARs, and infrared sensors to gather data for each task, which is then processed by high-speed parallel processors (FPGAs, GPUs, ASICs, and multi-threaded CPUs) for efficient execution and command transmission. Given the broad nature of the A/DAS market and

the complexity of these systems, researchers often opt for application-specific integrated circuit designs and ML algorithms with parallel processing capabilities to accelerate hardware performance and implement each task efficiently.

Advances in semiconductor technology and parallel processing have facilitated the widespread application of ML in this domain, leading to the development of new algorithms such as ANNs and advanced CNNs. The efficacy of ML algorithms is evidenced by various studies [11, 86, 114–119]. Performance metrics such as accuracy, sensitivity, flexibility, delay, and energy consumption are crucial in evaluating ML algorithm-based hardware and software applications. Analysis of ML algorithms in A/DAS systems and various hardware acceleration platforms considers these performance metrics, enabling comparisons and predictions for future work.

In this context, I include an analysis of the Performance Index (PI) as in (6) to demonstrate the price-performance metrics of the studies conducted.

$$PI = \frac{F \times T \times S}{C} \tag{6}$$

Where; C=Hardware Cost: Lower values are better. S=Image Size: Typically, larger values indicate higher resolution and potentially better detection capabilities. T=Number of A/DAS Tasks: Higher values are better, indicating more functionalities. F=FPS: Higher values are better, indicating smoother and potentially more reliable performance.

This analytical approach allows for a detailed comparative assessment of the output performance of the hardware platforms engaged in the research. To illustrate these advancements briefly, I offer an extensive comparative analysis in Table 1.4. This comparison elucidates the shortcomings of related works and demonstrates how the proposed methodology thoroughly addresses these challenges, thereby establishing a new standard for future A/DAS research.

The study conducted by Peng et al. [113] bears close relevance to this research. They developed a multitasking algorithm for A/DAS utilizing AMD Xilinx ZU9 hardware, successfully executing segmentation and multi-object detection tasks using the Cityscape dataset. Peng et al. assert improvements in inference performance through software and hardware optimizations. However, their software strategies appear conventional, relying

Table 1.4: Advancements in A/DAS: A comprehensive exploration of multi-learning frameworks, hardware software co-design, and diverse hardware accelerators for multi-task applications

Ref.	Depth est.	Drivable area & Line det.	Semantic seg.	Multi object det.	Dataset	Hw-Sw co-design	Platform	Image size	FPS	Evaluation (%)	PI (1e5)
[113]	N	Y	Y	Y	Cityscape	Y	AMD Xilinx ZU9	512×256	55	N (mAP) 57.59 (mIoU)	0.145
[120]	N	Y	N	N	FDDB	Y	C-based GPU Omap4460	640×480	18.2	87.3 (Accu.)	0.44
[121]	N	Y	N	N	COCO	N	NVIDIA Jetson Nano	320×320	20	60 (Accu.)	0.13
[122]	N	Y	N	N	Cityscapes KITTI	N	NVIDIA RTX TITAN	720×960	N	72.32 (mIoU)	N
[123]	N	Y	Y	N	tuSimple Cityscapes BDD100K	N	NVIDIA RTX 2080Ti GPU	416×416	N	95.73 (Accu.) 70.9 (F1) 57.03 (Accu.)	N
[124]	N	Y	N	N	Their	N	GTX 1080Ti Xavier NX	512×384	50 15	94.51 (F1)	0.078
[125]	N	Y	Y	N	Cityscapes	N	NVIDIA RTX 2080Ti & ROS	640×360	N	N	N
[126]	N	Y	Y	N	KITTI	N	NVIDIA TITAN GPU	224×224	N	94.21 (AP)	N
[127]	N	N	N	Y	VOT VBT	Y	Ultra96-V2	320×320	53.3	66.25 (mIoU)	0.18
[128]	N	N	N	Y	Their	N	NVIDIA GTX 1060	416×416	74.36	0.923 (mAP)	0.276
[129]	N	N	N	Y	CVC-09 FLIR A/DAS OSU, KAIST	N	NVIDIA Xavier AGX	640×640	43.59	84.1 (mAP)	0.21
[130]	N	Y	N	Y	Air learning database	Y	Xavier NX Jetson TX2	640×640	6	91 (Accu.)	0.098
[131]	N	N	Y	N	Cityscapes	N	GTX TITAN X Maxwell (GPU)	512×256	30	70 (mIoU)	0.14
[132]	N	Y	Y	Y	N	N	NVIDIA Jetson Xavier & TI TDA2x		10 15	39.78 (mAP)	N
[133]	N	N	Y	Y	BDD100K KITTI	N	Jetson Xavier	640×640	33.3	69.7 (mAP)	0.185
[134]	N	N	Y	Y	BDD100K+ Waymo CityScape KITTI	N	NVIDIA V100	320×512	N	43.7 (mAP) 46.2 (mIoU)	N
[135]	N	N	Y	Y	Woodscape	Y	SoC	572×572	10.5	67.76 (mAP) 62.6 (mIoU)	0.229
[136]	N	N	Y	Y	BDD100K	Y	KV260	288×512	37.49	N	0.442
[137]	N	Y	Y	Y	Cityscape	Y	AMD Xilinx ZU9	512×256	16.2	62.8 (mAP) 57.6 (mIoU)	0.0428
[138]	N	Y	Y	Y	N	N	Jetson Xavier TI TDA2X	512×256 512×256	10 15	39.78 (mAP) 70.17(mIoU)	0.119
[139]	N	Y	Y	Y	BDD100K CityScape KITTI Waymo	Y	KV260	320×512	25.4	Object det.: 51 (mAP) Segment.: 56.62 (mIoU) Drivable: 81.56 (mIoU) Line det.: 43.86 (IoU)	0.668
Multi-learning framework (Proposed)	Y	Y	Y	Y	BDD100K CityScape KITTI Waymo	Y	KV260	320×512	22.45	Depth Est.: 9.78 (SILog) Object det.: 50.06 (mAP) Segment.: 57.05 (mIoU) Drivable: 81.63 (mIoU) Line det.: 43.76 (IoU)	**0.738**

* Y/N: Yes/No, Accu.: Accuracy, F1: harmonic mean of the precision and recall scores, m/IoU: mean/Intersection over Union, m/AP: mean/Average Precision, FPS: Frame per Second, SILog: Scale invariant logarithmic error. The prices for the hardware platforms were sourced from various online platforms to ensure up-to-date and market-reflective valuations.

on standard multi-core CPU and DPU configurations without providing detailed implementations for hardware optimizations, particularly concerning on-chip memory (OCM) management. The Xilinx ZU9 platform crucially distinguishes between BRAM and URAM usage. BRAM is typically used for low-volume convolution, fast mathematical calculations, data processing, and reading/writing tasks. However, URAM is better suited for performing similar tasks but on larger matrices and tensors. Hence, Peng et al. must

address this distinction appropriately because it significantly affects computational efficiency. Their hardware optimization is limited, primarily focusing on reducing resource consumption through simpler operations without thoroughly analyzing their impact on performance and power consumption. Notably, they need to pay more attention to considerations regarding DPU frequency. In contrast, this thesis introduces a comprehensive optimization approach integrating advanced resource allocation, pipeline architecture, efficient BRAM, URAM utilization, and an optimized multitasking software model. This integrated approach has effectively addressed identified bottlenecks, enabling us to execute a broader range of A/DAS tasks and achieving a substantial 52.8% reduction in power consumption at a DPU frequency of 275 MHz, showcasing a significant improvement over Peng et al.'s methodology.

In their insightful work, Ghorbel et al., [120], proposed a method that utilizes GPU acceleration to parallelize the eyes detection algorithm based on Viola and Jones, a development aimed at designing an innovative smart wheelchair. The authors subsequently apply this research to craft a human-machine interface to govern intelligent wheelchair control. Notably, their work incorporates a significant element of software-hardware co-design. However, their study appears to lack comprehensive coverage of multi-task learning, which warrants further exploration to augment the design and performance of intelligent wheelchairs. In the concluding remarks of their study, Ghorbel et al. explicitly acknowledge the persistent challenge posed by GPUs in the context of energy consumption and efficiency. This issue is particularly pronounced in electronic systems powered by batteries, potentially limiting their research's viability and practical applicability in real-world settings. Moreover, the study employs the Omap4 4460 platform, which may be prohibitive considering the price-performance ratio. It is crucial to examine whether more cost-effective alternatives could achieve comparable, if not superior, results in designing intelligent wheelchairs and their respective human-machine interfaces. This would improve the technology's accessibility and affordability for a broader user base.

An additional noteworthy study is the one conducted by Cho et al., [126], titled "Multi-task Self-supervised Visual Representation Learning for Monocular Road Segmentation," which attained a commendable 94.23% AP performance. The authors employed a multi-task framework for segmentation-based roadway detection within their research. Their

study used the KITTI dataset and the considerably costly NVIDIA TITAN GPU as their hardware. In a distinctive shift from traditional literature, the authors examined the utility of unsupervised stereo-based indicators to acquire high-level semantic knowledge for monocular route detection. Their experimental outcomes indicated an above-average performance, albeit not significantly superior to our research. This thesis boasts several advantages over Cho et al.'s research, such as a broader array of utilized datasets (KITTI, BDD100K, Waymo, and CityScapes), the adopted methodology, the concurrent execution of five distinct tasks, and the incorporation of backbone encoder.

The authors of a study [127] conducted a comparative analysis of three different hardware accelerator methods for DNN. These methods included coarse-grained, fine-grained, and sequential Vitis-AI strategies. They developed two bespoke DNN architectures using the System Verilog and FINN frameworks, which showed the flexibility and applicability of these models. Although the authors achieved high performance in terms of FPS rate, they did not explore multi-task implementation in their work, which is an area for further investigation to improve computational efficiency. One critical observation about the study is the authors' choice of the high-performance MPSoC ZCU3EG for exclusive object detection tasks. Given this particular system's processing capabilities, it could be better utilized by distributing the computational load across multiple tasks, optimizing resource use. The study's single-task focus leaves the potential for more comprehensive, multi-task approaches largely unexplored. Such methodologies are beneficial for enhancing processing efficiency and performance in future research endeavors.

In their seminal study, Krishnan et al., [132], conducted an exhaustive analysis of domain-specific SoC design for autonomous vehicles, focusing specifically on Unmanned Aerial Vehicles (UAVs). Utilizing Xavier NX and Jetson TX2, they developed SoC accelerators tailored for nano, micro, and mini-UAVs, achieving acceleration factors of 2.25x, 1.62x, and 1.43x, respectively. Despite Krishnan et al.'s success, their methodology differs from this study in a significant aspect: software-hardware co-design, which is central to this research. This study further incorporates semantic segmentation, markedly enhancing FPS rates and providing a comparative advantage. Additionally, while Krishnan et al. preferred GPU-based accelerators known for high energy consumption due to extensive CUDA cores, this thesis addresses this inefficiency, underscoring our approach's unique-

ness and potential benefits.

Dellinger et al. [135] address challenges in deploying camera-based DL algorithms in resource-limited Automated Driving systems. Their proposed methodology combines a streamlined multitasking CNN with a low-power SoC, complemented by four fish-eye cameras. While emphasizing architectural optimization, CNN compression through pruning techniques, and enhanced real-time operability, their primary focus on pruning model parameters without exploring alternative optimization avenues and the ambiguity surrounding the "low power" designation of the SoC necessitates clarity and broader optimization considerations in their research.

Additionally, the adaptability of SoC architectures for integrated Hw-Sw co-design needs to be adequately explored. The absence of comprehensive optimization strategies encompassing task allocation, memory management, and the utilization of multiple CPUs stands out as a significant limitation within the study. The diminished performance outcomes further compound this despite a reduction in parameter count achieved through pruning. In contrast, this study showcases superior performance metrics utilizing an 8-bit-width quantized model, achieving notable enhancements in inference and accuracy. It is imperative to fully leverage the capabilities offered by multi-threading and Hw-Sw co-design, including both BRAM and URAM for OCM allocation, to rectify the identified deficiencies and elevate the efficacy of their proposed solution.

Fang et al. [137] present a novel real-time Aristotle architecture on the Xilinx ZU9 FPGA platform, featuring a unified shared backbone facilitator and subnetworks dedicated to segmentation and object detection. While the authors claim their framework to be energy-efficient and straightforward in development, our comparative analysis shows that their system incurs approximately 74% more energy consumption than our proposed methodology despite addressing a similar scope of A/DAS tasks. Furthermore, their focus on optimizations at the DPU level neglects potential enhancements accessible through alternative branches of the board.

Furthermore, despite their assertion of a Hw-Sw co-design approach, evidence suggests limited resource utilization, such as engaging only one out of four CPUs on the PS side and needing more multi-threading capabilities. These limitations may contribute to inconclusive outcomes. Fang et al.'s architecture also exhibits an inference speed 62.5%

slower than our study's. In summary, while their study contributes to the discourse on FPGA-based architectures for segmentation and object recognition, it needs substantial refinements to address inefficiencies and limitations for practical applications.

This thesis aligns with the methodology proposed by Lai et al. in their Multitask Semantic Attention Network (MTSAN) study [138], which integrates segmentation and object detection to enhance real-time A/DAS functionality while aiming to reduce false alarms. However, there is a crucial area for improvement in the reported frame rate by [138], as their use of development boards equipped with high-performance CUDA cores requires reevaluating the efficiency and variety of A/DAS tasks and performance benchmarks, deviating from our experimental findings. Moreover, although [138] introduces a Semantic Attention Module (SAM) to reduce computational expenses and enhance detection capabilities, their study needs comprehensive results. It presents deficient PI metrics, as Table 1.4 indicates. Additionally, the substantial energy demand of GPU-based boards makes them unsuitable for battery-operated devices, highlighting significant areas for methodological enhancement in their research.

It showcases superior real-time performance, exhibiting an improved FPS rate compared to [132, 138]. Furthermore, this thesis's mAP and mIoU results exceed the MTSAN model's results, underscoring our superior performance in object detection and segmentation tasks.

Regarding power consumption, our model operates on an MPSoC architecture, renowned for delivering robust computational performance while consuming significantly less power than GPU-based architectures. This issue summarises the model being more energy-efficient while providing full A/DAS functionalities. Regarding memory resources, this research leverage the Xilinx Kria KV260 platform, renowned for optimal memory resource utilization, which results in a more memory-efficient solution.

In summary, this model presents a more balanced solution for A/DAS applications by providing robust performance coupled with energy efficiency and cost-effectiveness. This situation culminates in the model outperforming those proposed by [132, 138] across multiple evaluation parameters, asserting its superiority in this field.

Masayuki Miyama's research [140] offers a multitasking CNN for driving assistance, handling object detection, semantic segmentation, and disparity estimation. However, its

robustness could be better due to the automatic loss of weight adjustment and the unproven claim that decoder simplification boosts accuracy. The study's neglect of Hw-Sw co-design and sole reliance on the CityScapes dataset limit its real-world applicability and optimization potential. Conversely, this approach uses four datasets and incorporates Hw-Sw co-design, enhancing performance and scalability for real-world scenarios.

As a result, Table 1.4 highlights crucial A/DAS parameters, including Hw-Sw co-design, image size, FPS, accuracy, and performance index. This study contributes significantly to the literature by exploring uncharted territory, resulting in higher PI values.

1.4 Thesis Statement and Contributions

To ensure this thesis's reliability and cutting-edge nature, approximately 300 international studies were analyzed. This comprehensive review allowed for detailed comparative analyses across various studies, including "Traffic Sign Detection," "Road and Lane Detection," "Vehicle and Pedestrian Detection," and "Driver Behavior," all of which involve multiple A/DAS tasks. These analyses were systematically presented in tables, shedding light on previously unexplored aspects within the literature and identifying areas requiring further investigation.

The research primarily delves into the development of innovative multi-tasking learning models provided by AMD Xilinx and their implementation on Custom Deep Learning Processing Units (DPUs). The principal aim is to develop a cost-effective architecture that paves the way for the simultaneous execution of multiple A/DAS tasks on a single hardware platform. This involves the concurrent development of processing systems (PS), programmable logic (PL), and processing engines (PE) to optimize performance, minimize power consumption, and manage memory constraints effectively. The thesis encompasses three distinct applications, each employing Custom DPUs tailored explicitly to individual studies. It also includes an analysis of the frequency and resource consumption of the designed DPUs. The contributions of this work to the academic field are substantial, particularly in demonstrating a state-of-the-art impact on the development of intelligent vehicles and A/DAS applications. The specific scholarly contributions of this thesis are outlined below:

1. A unique custom overlay with a DPU designed specifically for the A/DAS application has been developed. This original design enhances computational efficiency and optimizes hardware resources, providing a robust foundation for the research.

2. Int-8-bit width model quantization was employed to reduce computation load and memory usage, optimizing hardware resource utilization. This approach maintains model accuracy while enhancing processing efficiency.

3. A detailed, multitasking A/DAS application was executed on the Custom DPU, encompassing a range of functions such as segmentation, multi-object detection, line detection, drivable area segmentation, and depth estimation. This approach

offers a thorough and flexible solution that meets the varied requirements of A/DAS applications.

4. A Hw-Sw co-design methodology was utilized to integrate a custom DPU with software components and enable collaborative processing. This strategy streamlined development, optimized system performance, and efficiently utilized hardware and software resources for improved efficiency.

5. The pipeline structure was enhanced to boost performance and minimize response time. This optimization facilitates the efficient execution of intricate DL models, enhancing real-time responsiveness in A/DAS applications.

6. Due to the various hardware platforms, software packages, DL models, and image sizes used in A/DAS literature, making accurate comparisons between studies is challenging. The PI criterion, which is absent in the current literature, was introduced to address this issue and enable comprehensive performance comparisons. This criterion lays the basis for comprehensive comparisons among all studies.

7. Frequency analysis was conducted to investigate the energy efficiency of all Custom Xilinx DPU-TRDs for A/DAS multi-task learning applications. This research effort fulfilled the existing scholarly gap in frequency-dependent resource and energy consumption analysis.

8. In addition, some special applications performance comparisons were conducted on the PYNQ-Z1 development board by using several framework like FINN and Python productivity PYNQ to guide interest.

1.5 Thesis Organization

The organization of the thesis is as follows: Section 1 identifies the key motivations behind the research, primarily focusing on the technological advancements that can significantly enhance the safety and efficiency of A/DAS. A detailed literature review then systematically explores existing studies, identifying trends and significant gaps in A/DAS applications. This thesis presents novel contributions to the field by introducing innovative methods and findings. The section concludes by providing readers with a clear layout of the thesis structure, offering a roadmap that outlines the scope and sequence of the topics discussed.

Section 3 dives deep into the role of ML tools that enhance the functionality and reliability of embedded platforms used in A/DAS. This section discusses core applications of A/DAS, including traffic signs, road and lane detection, and detection of vehicles and pedestrians, all critical to automotive safety. Additionally, the capabilities for handling multiple A/DAS tasks concurrently are examined, showcasing the integration and synchronization required for high-performance systems. Challenges specific to these applications are thoroughly analyzed, covering aspects such as power consumption, execution time, accuracy, memory usage, scalability, flexibility, and potential bottlenecks. The discussion concludes by delving into various optimization techniques that target hardware and software enhancements, thereby improving embedded systems' overall efficiency and performance in A/DAS.

Section 4 discusses how to strategically use hardware acceleration to improve embedded system performance, with a focus on the PYNQ-Z1 platform, among others. This section begins by exploring the acceleration of real-time video processing for edge detection on the PYNQ-Z1, illustrating how this application significantly benefits from hardware acceleration in terms of processing speed and efficiency. Further, the section details the implementation of FIR filters on the PYNQ-Z1, where the hardware and software design are thoroughly examined. The testing and evaluation methodology is also discussed, providing insights into the results that underscore the effectiveness of hardware acceleration in reducing computational load and improving response times. In addition to these specific case studies, the section covers other practical applications facilitated by the PYNQ-Z1 platform. These include the development of custom NN models optimized for low-precision operations, which are crucial for tasks like object recognition and

classification in real-time systems. Each step, from setup and training to implementation and performance analysis, is elaborated upon, demonstrating the platform's capabilities in handling complex computational tasks efficiently. This section concludes by synthesizing the findings from various applications, emphasizing the crucial role of hardware accelerators like the PYNQ-Z1 in modern embedded systems. The results showcase system performance improvements and suggest future research and development directions that could further exploit the advantages of hardware acceleration.

Section 5 explores the comprehensive integration of the AMD Xilinx Vitis-AI framework, a critical tool for implementing advanced ML models in A/DAS. This section provides an in-depth overview of the framework's capabilities, starting with the general architecture and setup and moving into specific applications in multi-task learning environments suitable for automotive systems. The section begins by outlining the general framework 5.1 of A/DAS multi-task applications, highlighting the adaptability and scalability of the Vitis-AI environment. It details the setup process involving the DPU-TRD, a crucial component in configuring the hardware to run AI models efficiently. This discussion includes the initial design phases using Vivado 2022.1 IDE and the subsequent deployment of these designs using PetaLinux 2022.1, providing insights into the seamless hardware and software integration. Following the setup, the section examines the specifics of the hardware configuration, detailing the optimization processes that ensure maximum efficiency and throughput in real-time applications. This is complemented by examining the software configuration strategies that align with the hardware capabilities to enhance performance further. This section discusses the configuration and optimization of the multi-learning model 5.1.2 applications within Vitis-AI. It explores the shared backbone encoder for feature extraction, which serves multiple heads for segmentation, detection, and depth estimation tasks. Each of these components is essential for comprehensive scene understanding in A/DAS, and their implementations are discussed in detail, including the challenges and solutions encountered during development. QAT 5.1.3 is another focal point in this section. It addresses the need to maintain model accuracy while reducing computational demands. This technique ensures that the models deployed on the hardware are efficient and effective, meeting the severe requirements of real-time automotive applications. Furthermore, the section includes a thorough discussion of the datasets used

for training these models, the training processes themselves, and the subsequent model evaluations. These discussions highlight the iterative model refinement and optimization process to meet the exacting standards of automotive safety and functionality. Finally, it discusses the significant improvements in processing speeds, reduction in power consumption, and enhanced accuracy of detection and prediction models facilitated by the framework.

Section 6 brings together the research findings and discusses this work's senses and potential impact on the automotive industry. This section evaluates the advancements made through the research and highlights how these contribute to the field of embedded systems and driver assistance technologies. It offers a comprehensive description that emphasizes the depth and thorough nature of the research, aiming to provide significant insights and advancements to these fields.

Following the conclusions drawn, Section 7 outlines prospective avenues of research and development this thesis suggests. It proposes future studies that could further enhance the functionalities and applications of A/DAS, encouraging exploration of new developments that could propel progress in this critical area.

2 Technical Background of Neural Network Computation

This section offers a detailed examination of the computational foundations of NNs, emphasizing how these operations are implemented on various hardware accelerators. Understanding these computational principles is essential for optimizing the performance and efficiency of NN applications in real-time systems such as ADAS.

2.1 Matrix-Vector Operations

NNs rely on matrix-vector operations combined with activation functions to transform input data into output decisions. Each layer of a NN typically computes an output $\mathbf{y}$ using the formula:

$$\mathbf{y} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

where $\mathbf{W}$ is a matrix of weights, $\mathbf{x}$ is an input vector, $\mathbf{b}$ is a bias vector, and σ represents a nonlinear activation function such as ReLU or Sigmoid. These computations are mathematically intensive and involve significant data movement, which can be a bottleneck in hardware implementation. Efficient execution on hardware platforms like ASICs, FPGAs, and GPUs typically involves optimizing these operations to utilize parallel processing capabilities and minimize memory fetch times. Understanding these dynamics is critical for engineers designing hardware-accelerated NN systems.

2.2 Optimization Algorithms

To enhance the performance and efficiency of NNs, especially in constrained environments like embedded systems, various optimization techniques are applied. These include:

- **Network Pruning:** Reducing the complexity of the model by removing weights that have little impact on the output, effectively decreasing the size and computational demands of the network.
- **Quantization:** Lowering the precision of the numerical values used in computations reduces both the memory requirements and the computational load. The

quantization process can be expressed mathematically as:

$$\text{Quantized value: } \lfloor \frac{weight}{scale} \rceil$$

This technique is particularly beneficial for deployment on hardware accelerators where memory bandwidth and power consumption are limited.

These optimizations not only improve the execution speed but also enhance the energy efficiency of NN, making them more suitable for real-time applications.

2.3 Software Environment

The development and deployment of NNs on hardware accelerators are supported by specialized software environments that provide tools and frameworks designed to optimize computational efficiency. These environments include platforms like TensorFlow, PyTorch, and others that support CUDA for GPUs or HLS tools for FPGAs. These tools offer functionalities such as automatic optimization of matrix operations, integration of quantization, and efficient memory management strategies. They allow developers to focus on model design and algorithm development, abstracting away many lower-level hardware considerations.

3 Recent Advances in Embedded Systems

This section explores the latest advancements in A/DAS within embedded platforms, beginning with an exploration of A/DAS's definition, purpose, required components, and the prerequisites for its proper implementation. Additionally, it provides a comprehensive coverage of the hardware platforms and software methodologies used in A/DAS.

3.1 ML Tools for Embedded Platforms

Developing an application with high accuracy and performance is tough since ML development requires theoretical foundations. However, the growing ML ecosystem has made it accessible to embedded system developers who want to develop ML by reducing the need for a deep understanding of the algorithms used in this ecosystem [141]. What can be seen in Table 3.1 is the most popular ML software tools from the past to the present. Creating correct inference models from ML tools used in embedded platforms is one of the outcomes of a learning process that can utilize a wide variety of basic model types and architectures. Among these model types, ANNs, or more advanced CNNs, quickly gained popularity in A/DAS systems due to their success in image recognition, environmental management, and classification. Before moving on to the studies in this direction in the literature, I have present CNN models & computational challenges in Table 3.2 and Toolflow characteristics in Table 3.3. VGG16 & VGG19, introduced by Zisserman and Simon Yan in 2014 [142], is one of the first CNN models to emerge. It is a simple structure using tiny 3x3 dimensional convolution filters and blocks of increasing convolution layers in depth. The study includes examining the effect of convolutional network depth on large-scale image recognition setting accuracy.

ResNet architecture was developed by He et al. in 2015 [143], was a milestone in offering a type of architecture based on "modules" or "networks within networks." Inception-V4 architecture was developed by Szegedy et al. in 2014 [144], offers improved use of computing resources within the network. Xception, proposed by François Chollet (creator of Keras) in 2016 [145], is an architecture that reduces convolutional computing times. It performs better than Inception V3 on larger image classification data sets. This model uses different-sized filter blocks that can be combined later to identify features of different

Table 3.1: A List of ml frameworks available for use in specific applications [11–17]

ML Library	Win. Mac. Linux	Open Source	C/C++, Python	Algorithms or Features	in A/DAS
PyTorch	✓	✓	✓	Optim Module NN Module Autograd Module	✓
TensorFlow	✓	✓	✓	Provides a library for data-flow programming	✓
Knime	✓	✓	Java	Data manipulation Data mining	✓
Weka	✓	✓	Java	Classification Clustering Regression Data preparation	✓
Scikit-Learn	✓	✓	✓	Classification Clustering Regression Data preparation Dimensionality reduction	✓
Shogun	✓	✓	C++	Classification Clustering Regression SVM Dimensionality Reduction	✓
Accord.Net	✓	✓	C#	Classification Clustering Regression Distribution Audition Computer Vision Kernel Methods Hypothesis Tests	✓
Apache Mahout	✓	✓	Java, Scala	Clustering Regression Preprocessing Data mining Math-based functions	N/A
RapidMiner	✓	N/A	Java	Data loading and Transformation Preprocessing and Visualization	N/A
Keras. io	✓	✓	Python	API for NNs	✓
Google Cloud AutoML	✓	N/A	C#, Java, Phyton	Text recognition Speech recognition	N/A
Colab	Cloud Service	✓	Phyton, Scala	Supports libraries of PyTorch Keras, TensorFlow OpenCV	✓
Jupyter Network	✓	✓	Python, R	Inpython navigator wAnaconda navigator	✓
Orenge3	✓	✓	✓	Preprocessing Data visualization Data mining	N/A
IBM Watson	Linux	N/A	C++, Java, Prolog	Natural language processing Information retrieval Knowledge representation Automated reasoning	N/A
Amazon ML (AML)	✓	N/A	✓	Multi-class classification Binary classification Regression	✓

scales. SqueezeNet, introduced by Iandola, Forrest N., and their colleagues in 2016 [145], aimed to create a small NN with fewer parameters that could easily fit into the computer memory. LeNet-5 architecture, which has two convolutional and three fully connected layers, consists of 5 layers, as can be deduced from the name. It is a convolutional NN

Table 3.2: CNN Models & computational challenges for specific applications

CNN Models	Year	Depth	Design Principles	Challenges
LeNet-5	1998	5	2 convolutional and 3 fully-connected layers	N/A
AlexNet	2012	8	Increased depth, Increased layer width	Non-uniform filter sizes Grouped convolutions
ZFNet	2013		Wider layers	Computational load Memory footprint
VGG16 & VGG19	2014	16	Increased depth Increased layer width	
GoogleNet		22	Inception module	Irregular computations Irregular layer connectivity
ResNet-152	2015	152	Residual block	
Inception-V3		25M. param	Avoid representational bottlenecks	N/A
YOLO		53	Single stage CNN, train end to end	Earlier version struggles to detect small object
Xception	2016	72	Depthwise Seperable Conv	N/A
SquezeeNet			Compresses	
Inception-V4		N/A	Residual Inception block	Irregular computations Irregular layer connectivity
DenseNet-161	2017	161	Dense block	

Table 3.3: ToolFlow features in specific applications

Framework	Library	NN Models
fpgaConvNet	Caffe & Torch	CNN, Res, Incep, Dense
DeepBurning	Caffe	CNN, RNN, DNN
Angel-Eye		CNN, DNN
ALAMO		
Haddoc2		
DnnWeaver		
Caffeine		
AutoCodeGen	Proprietary Input	
FINN	Theano	BNN
FP-DNN	TensorFlow	CNN, RNN, DNN, Res
Snowflake	Torch	CNN, Res, Incep
SystolicArrayArchitecture	C-Programming	
FFTCodGen	Proprietary Input	

proposed in 1989 by Yann Lecun et al. [146]. introduced by Alex Krizhevsky et al. in 2012 [147], AlexNet is a structure consisting of 8 layers in total with five convolutional and three fully connected layers. They were the first to use ReLUs as an activation function when it first appeared. GoogleNet or Inception-v1 captures and sets different properties with 1x1, 3x3, and 5x5 having parallel convolution layers with different filters. This idea was introduced by Arora et al. in 2014 [144].

Szegedy created Inception-v3, Christian et al. in 2016 [145] as the successor to Inception-v1. Inception-v2 is not used much because of its similarity and lower sophistication than Inception-v3. Improved aspects compared to Inception-v1: Factoring nxn convolutions into asymmetric convolutions as 1xn and nx1, factoring the 5x5 convolution structure with two 3x3 convolutions, and finally replacing the 7x7 convolution with 3x3 serial convolution. Providing convolutional networks with shorter connections between the layers close to the input and output has shown that it is significantly more profound, efficient, and

accurate for training. In light of these observations, DenseNet was introduced, a dense convolution network formed by connecting each layer to the other layers in a feed-forward manner [148].

3.2 A/DAS Applications

Nowadays, designers have increasingly focused on researching and developing specific aspects and challenges within the automotive industry. Key developments include autonomous vehicles (self-driving cars) [149], advancements in environmental and traffic management [150], and enhancements in in-vehicle communication systems [151]. These topics have been extensively discussed in the literature. Additionally, there exists a robust body of research explicitly targeting A/DAS [114, 152–155], often introducing new perspectives that either focus on particular aspects of A/DAS or encompass the entire spectrum of these systems.

After extensive research, our paper categorizes and analyzes various studies in a structured format. We begin by outlining the preferred types of hardware for each study and explaining the reasons behind these choices. We then provide a detailed overview of each study under distinct headings in various tables. Each section briefly describes each ADAS functionality, explaining the rationale for hardware selection.

- **Traffic Sign Detection**: Utilizes GPUs for their superior image processing capabilities, which are essential for the real-time detection and classification of traffic signs. FPGAs are also employed for their flexibility in handling varied image processing algorithms at high speeds.

- **Road & Lane Detection**: Often implemented on FPGAs and ASICs, these applications benefit from the low-latency and high-efficiency processing necessary for real-time lane marking recognition and vehicle positioning.

- **Vehicle & Pedestrian Detection**: This crucial safety feature leverages the computational power of FPGAs and GPUs to perform complex object detection and tracking algorithms. ASICs are also used for their optimized power and efficiency in handling these specific tasks.

- **Driver's Behaviours**: Analyzing driver behavior involves sensors and data analytics, where low power-consuming FPGAs and ASICs are often preferred. GPUs may also support more intensive behavioral analysis tasks, such as fatigue detection through facial recognition algorithms.

- **Including More Than One A/DAS Task**: Hybrid systems that integrate CPUs, GPUs, and FPGAs offer an adaptable platform capable of handling multiple ADAS functionalities simultaneously. This strategic approach ensures that the computational load is distributed efficiently, thereby optimizing the overall system performance and responsiveness, a crucial aspect in the realm of ADAS development and implementation.

Following these initial descriptions, detailed explanations and tables explore each ADAS functionality and its associated studies.

3.2.1 Traffic Sign Detection

Traffic signs consist of prohibitive or permissive signs placed by officials in certain places to adjust driver behavior while the vehicle is in motion. Drivers must act according to these signs for a safe ride. However, in some cases, it is inevitable for the driver to miss these signs and cause an accident due to distraction or environmental factors. Traffic sign recognition (TSR) systems and GPS have been developed to prevent these behaviors. In this way, the driver's distraction is eliminated or partially reduced through the TSR and GPS. These two methods mentioned may be impossible in some cases. While GPS information loses effect in tunnels or mountainous areas, designers must adjust delay and accuracy well since TSR systems are computer-based.

It is challenging to make a fair comparison between the studies. TSR studies in the literature have many different platforms, frameworks, CNN models, and algorithms. As can be understood from the literature studies given above, while smartphones, ready-to-use frameworks, libraries, etc., are used in some studies, some of them, embedded systems with parallel processors, existing data sets, etc., are utilized. In light of this information, I have created Table 3.4 and Table 3.5, which I evaluate according to performance parameters by examining the studies in which similar hardware and software, similar frameworks,

Table 3.4: Performance comparison of A/DAS traffic sign & light detection task in the literature

Ref.	Mission	Detection Methods	Classification	Datasets	Accuracy(%)
[156]	26 traffic signs	YOLO-V4 based CSP-DarkNet53	R-CNN	1390 (own)	74.91
[10]	Traffic signs	CNN, DNN	DCGAN, WGAN	GTSRB	N/A
[157]	Prohibitory, Danger and Mandatory signs	VGG16 & InceptionV2	HoG	37079 (own)	89.4 73.2 Recall
[158]	35 traffic signs	MSER	LVSM, HoG	GTSBR	96
[159]	43 traffic signs	MB-LBP+CNN	CNN	GTSBR &BTSD	94.25
[160]	Green & Red lights	HSL, Shape rations	Hog-LBP, K-ELM+FSM	Own	93.2
[161]	25 traffic lights	N/A	SURF & SVM	225 (own)	96
[162]	43 traffic signs	YOLOv3 MobileNetv2	R-CNN	GTSBR/DB	N/A
[163]	50 traffic signs	Color Segmentation Thresholding Shape matching	SVM	Own	90.1
[164]	500 traffic signs	LBP, Shape matching Color Segmentation	Hough trans. method	Own	80
[165]	Pedestrian Vehicles Traffic signs	Scale-invariant feature transform	AdaBoost & CNN	TI automotive	84.6
[166]	Traffic lights	Blob detection	HoG, SVM	Own	90
[167]	Traffic signs	Gray scale	VGG16 based CNN, Post-training quantization	GTSDB	99.2
[168]	Prohibitory signs Danger signs Mandatory signs	Thresholding Filtering & Labeling Matching	Shape classification A template-based correlation method	GTSDB Own	97.39 96.87
[169]	Traffic signs	N/A	Bracnh CNN	GTSRB	99.12
[170]	Traffic signs	Color segmentation Speed-up robust features	k-NN, k-Means	Own	97.67
[171]	Road signs	The MSER region detector Labeling, Filtering RoI extraction	N/A	GRAM,GTSDB	88.94
[172]	Prohibitory signs Danger signs Mandatory signs	Normalized RGB color transform CCL	HoG, SVM	GTSDB	Det. 96.61 Rec. 90.85
[173]	Traffic sign rec.	Color segmentation Hue detection Labeling Morphological filtering	k-NN Feature extraction	Illinois road signs	97.54
[174]	10 speed limit signs	Contour detection Thresholding HSV, CCL	SVM, CtC	Own	95
[175]	9 speed sign bundles	Gray scale Cascade filter Haar features	Adaptive thresholding Size filtering %OPF	Own	99.65
[176]	Traffic signs	SVM, HOG	CNN	GTSRB	98
[177]	Traffic signs	Yolo v4, Yolov4-Tiny	CNN, Faster R-CNN	TWTSD	99.4

and similar CNN models were used in recent years.

As seen in Tables 3.4 and 3.5, there are distinguishing parameters such as method, classification, hardware, cost per pixel (7), data set, and algorithm in the recognition and detection of traffic signs/lights. So, I can deduce that designers prefer to use embedded platforms in such applications instead of personal computers. As seen in Table 3.5, smartphones are better than others in traffic sign/light detection applications concerning price/performance. But here, when we consider the TS/LD part of the A/DAS system, we must take into other

Table 3.5: A/DAS tsd implementation information in the literature

Ref.	Libraries-Language	Hardware-Platform	Image size	FPS	Cpp ($\times 10^{-5}$\$)
[156]	Pascal VOC	Embedded Google-Colab Tesla K80 GPU-12GB RAM	720p	18	2.41
[157]	Caffe, SNPE SDK	Embedded Qualcomm Snapdragon TM 820A with GPU	1280x720	7	6.2
[178]	TensorFlow	Embedded NVIDIA Jetson TX1	640x480	14.9	6.53
[179]	Keras	Embedded NVIDIA Jetson TK1-TX1	640x480	19 - 30	3.42-3.24
[165]	TI's Vision SDK	Embedded TI Low power TDA3x SoC ARM Cortex M4	32x32	30	0.052
[166]	N/A	Embedded FPGA based SoC Xilinx Zynq ZC-702	1024x768 XGA	60	1.89
[167]	TensorFlow	Embedded FPGA combined with host processor-based edge device	32x32	N/A	N/A
[168]	OpenCV	Embedded ARM dual-core ZedBoard Zynq-7020	1360x800	11	4.92
[170]	OpenCV, C/C++	Embedded ARM Cortex-A9 1GHz Zynq-7000 FPGA	320x240	1.01	385.4
[171]	OpenCV, C++	Embedded ARM 800MHz Xilinx Zynq SoC ZedBoard	1360x800	12	4.511
[174]	OpenCV	Embedded; Asus PF500KL RadXA Rock Pro	640x480	44 - 32	1.84-1.22
[163]	OpenCV/Qt-library	Embedded FriendlyARM Tiny4412	640x480	15	2.17
[158]	N/A	Smart Phone IPhone 6s Dual-core 1.84GHz A9	640x480	30	4.036
[159]	TensorFlow	SmartPhone Huawei P9 Lite ARM-Cortex 53 8-Cores	640x480	10	7.42
[160]	Open CL	Smart Phone Samsung Note-3 CPU–GPU fusion-based	1280x720	20	1.10
[161]	N/A	SmartPhone Quad-Core 2.5GHz	500x400	0.251	498
[164]	N/A	SmartPhones; Samsung s4 Apple 5s	320x180	N/A	N/A
[175]	C	SmartPhone 2.5GHz 2GB RAM Android	35x35	25	326
[162]	Pascal VOC0	NVIDIA TITAN Xp GPU Intel(R) i7-8700 CPU 3.2GHz	800x1300	18	5.89
[169]	BCNN	CPU E5-2620 2.1GHz	15x15, 222x193	N/A	N/A
[172]	OpenCV, C++	PC Intel Xeon E3-1230v2 3.2GHz CPU 16GB RAM	1360x800	135	7.48

embedded platforms since we will not implement all other A/DAS applications. These applications include road/lane detection, vehicle/pedestrian detection, and driver behavior. Since numerous embedded platforms exist, one must consider performance, cost, power consumption, and footprint criteria when selecting the target hardware. As a result of this research, designers can choose FPGA-based SoC platforms based on their reconfigurable capability, price performance criteria, ease of use, and the ability to perform parallel processing.

$$Cpp = \frac{Hardware\ Cost}{(Image\ Size) \times (FPS)} \tag{7}$$

3.2.2 Road & Lane Detection

Traffic road and lane detection (TRLD) are also part of the A/DAS, like recognizing traffic lights and signs. Environmental conditions must be controlled in the same way

while performing TRLD tasks. While the vehicle moves in the designated lane or road, it is necessary to follow the peripheral movements to overtake by following the others in front or behind, accelerating and decelerating. Sensors are used to examine specified environmental activities. Each piece of information received from the sensors imposes a separate calculation load. Since the obtained data is always on time and requires complex mathematical operations, it must be analyzed and processed quickly and accurately. Recently, researchers have adopted DL methods for processing this data. CNNs optimization is required for data with high memory requirements in real-time hardware applications. Studies involve various hardware, software, and libraries in which DL algorithms are used for TRLDs. I have recently examined this field's research and information about what to do.

The authors in [105] stated that pre-trained networks could be used for lane tracking, using DL models that can be used when there is no lane in the image. They mentioned that these methods are unsuccessful even in such cases and that a new approach is required to overcome this problem. The authors have proposed how each lane has its own example of training this end-to-end lane detection problem. They claimed to have verified competitive results on their simple data sets, with a throughput of 50FPS, capable of processing a variable number of lanes and controlling lane change situations. With the method they call LaneNet, the image is first encoded as an embedding branch and segmentation branch with the shared encoder. The pixel embeddings and binary lane segmentation features are extracted, and these features are collected as clustering. In this way, LaneNet is trained end-to-end so that the number of lanes the network can detect is not limited and can cope with a lane change. The authors compared the training and post-training behavior of two unlike networks, LaneNet and H-net experimental results. According to this comparison, the LaneNet network achieved a performance of 62.5FPS with a 512x256 size image. In contrast, the H-net network reached a speed of 41.66FPS (on paper, it is mentioned as 416.6 FPS for H-net, but the total FPS for both networks is given as 52.6, possibly 41.66 inadvertently misspelled as 416.6) with a 128x64 image. The authors should have provided information about the hardware platform, software language, accuracy, and framework.

The authors in [180] presented a stereo vision lane detection and classification system to obtain lateral positioning and collision warning of DL-based ego cars for A/DAS systems.

In A/DAS, the network was trained using a manually tagged ROI with the KITTI data sets to establish the lane change strategy and classify the computer's right/left sidelines. The proposed system comprises segmentation, IPM and Sobel filtering, hough transformation, lane detection, and classification. The authors stated that they obtained 99.6% accuracy from their study with data sets consisting of low-noise bird's-eye views of 1280x400 pixels and 22000 images. They used the GTX-1080TI to train the network for about 5 hours. The authors also used their own data sets to evaluate the method's performance and showed that they obtained results similar to the KITTI data set in the table. They utilized an Intel Core i7-3.1GHz, a 4-core processor with NVIDIA GeForce GTX650 and 4GB of RAM to test the entire system.

The authors in [181] stated that they made inferences using old data on roads with bad weather, traffic congestion, and erased lanes, and they did this by using CNN and RNN together. They stated that they tried this in two large data sets and that the suggested method was more successful in lane detection under challenging conditions than their competitors. The proposed method is based on an encoder-decoder logic that takes more than one continuous-time image as input and predicts the current frame to divide it into segments semantically. In the recommended method, images for lane detection were sampled at 256x128 resolution and tested on two GeForce GTX TITAN-XGPUs with Intel Core Xeon E5-2630, 2.3 GHz, 64GB RAM. The authors stated that the accuracy of the method they proposed was 98%.

In [182], the authors have presented an ultra-low-power hardware accelerator for path detection. They stated that the structure in which the CNN model named RoadNet is used offers an incredible power consumption of 80mW and an approximately 300 GOP/s /W power efficiency. This structure has a processing capacity of 241FPS for a 1080P resolution image. The detection and segmentation of the road are difficult due to different weather conditions and difficulties caused by heavy shadows. For this reason, the authors stated that they could determine whether the pixels belong to the road without comparing them with the samples obtained by calculating the heat map of the road with the previously trained network. As a result, the authors achieved an accuracy of 96.76% and a recall performance of 89.58%. In addition, the authors highlighted the RoadNet (VLSI design) chip design as a 32 nm technology with a 500 MHz clock speed, an estimated size of

$0.45mm^2$, and a power consumption of 80mW.

In another embedded system study [183], the authors stated that they implemented an FPGA-based real-time hardware application using a stereo camera for autonomous vehicles and accordingly achieved good results in experimental studies. The authors also commented that if all operations were only software-based, there would be a delay of at least one frame and much less (about 10ms) when used with an FPGA-based hardware accelerator. The experimental results stated that it was 3x faster than the software method. The system, consisting of 4 cameras, front, rear, left, and right, was built on a Spartan-6 board (for image processing) using OpenCV dynamic programming and a stereo camera with an FPGA-based computing board (for processing real-time video signals). The stereo camera resolution used in the application is 5120x720 and 30FPS. The experimental results obtained 95.4% accuracy and 27FPS performance for lane recognition.

Real-time road segmentation is performed using LIDAR sensors. The authors in [184] stated that they used LIDAR to implement a CNN-based real-time FPGA (Xilinx UltraScale XCKU115 FPGA) with high accuracy. Their work consists of pre-processing, NN processing, and post-proc essing. The network structure consists of 5x5 kernels, 64 filter banks, and 11 convolutional layers, and they explained that they get a good trade-off between CNN performance and resource usage with this setting. They trained this network using KITTI lane detection and achieved an average accuracy rate of 84.76% and an execution-time success of 16.9ms. The authors also evaluated the algorithm on the K20 GPU using MATLAB Caffe and obtained the total processing time as 120ms and with the Intel Xeon CPU E5-2687W v3 as 500ms.

There are many studies on lane detection in the literature. As the research shows, more is needed to produce the optimum solution so far. I presented the studies on this subject in recent years to the reader in a table by considering the distinguishing factors such as the method used, the hardware and software environment, cost, and efficiency. As seen in Table 3.6, many platforms are based on road/lane detection hardware. It is challenging to make an exact comparison between these platforms. When we make price and performance analyses, the performance of GPU-based hardware stands out, while when power consumption, flexibility, and portability are considered, embedded systems (FPGA) stand out. GPU-based platforms are often used for network training. However, their high

Table 3.6: Performance comparison for A/DAS traffic road & lane detection tasks in the literature

Ref.	Mission	data-sets	Embedding Methods	Hardware	Image Size	Accuracy(%)	FPS	Cpp ($\times 10^{-5}$ \$)
[105]	Lane track.	tuSimple LaneNet H-Net	DL	N/A	512x256 128x64	N/A	50	N/A
[122]	Lane detect. &tracking	N/A	N/A	CPU	480x640	87-98	30	N/A
[180]	Lane detect.	KITTI	DL	i7-3.1GHz Geforce GTX650	1280x400	99.6	N/A	N/A
[182]	Road detect.	RoadNet Cityscapes KITTI	CNN	ASICs (own)	1080P	96.76	241	N/A
[181]	Road inference	SegNet U-Net	CNN,RNN	GTX TITAN Intel Core Xeon E5-2630	256x128	98	N/A	N/A
[185]	Lane Detect.	tuSimple SCNN LaneNet	HLS	NVIDIA Drive PX2 1080Ti	1280x720	96.53 96.4	23 50	2.38
[125]	Road detect.	Cityscapes	PyTorch	NVIDIA Rtx2080ti	784x784	82-89	N/A	N/A
[186]	Road detect.	Cityscapes KITTI	CNN	NVIDIA Titan Rtx	2048x1024 400x800	96.8	33.3	0.023
[121]	Road Peculiarities Detection	SSD-Mobile Net	TensorFlow	NVIDIA Jetson Nano	N/A	65	20	N/A
[183]	Lane recog.	N/A	OpenCV Dynamic Prog.	Spartan-6 & FPGA-based HW. accelerator	5120x720	95.4	27	0.104
[184]	Road Segmentation	KITTI	CNN	Xilinx Ultra-Scale XCKU115	256x64	84.76	59.17	0.0205
[187]	Lane detect.	Cifar10	Vivado Caffee, DL	Xilinx ZYNQ7035	320x480	N/A	104	6.26
[188]	Finding drivable free space	Cityscapes	Sparse CNN Layer fusion	TI TDA 3x SoC	1024x512	95.9	20	7.62

energy consumption has restricted their use. In addition to these, hybrid models are also preferred. While GPU accelerators are used for network training in these models, testing and running the system are done with embedded platforms.

3.2.3 Vehicle & Pedestrian Detection

Another critical area of A/DAS is vehicle and pedestrian detection. Vehicle and pedestrian detection usually consists of preprocessing, feature extraction, and classification. These start with preprocessing, such as ROI and thresholding, and then continue with feature extraction with algorithms such as HOG, CoHOG, and Haar. Finally, it will use classification algorithms such as SVM, K-nearest Neighbors, stochastic gradient descent (SGD), and logistic regression. The SVM classifier is generally preferred because of its low memory requirement and easy implementation on embedded platforms.

DL algorithms like CNNs in A/DAS systems might cause problems in embedded platforms with high memory requirements. Specific optimizations are applied to these networks to eliminate these problems. These steps are popular because pruning and quantizing are in widespread use. Even if these optimization methods are involved, there is still a problem in object detection and classification as there is a need for sequential tasks for each task [86]. Special networks such as YOLO [189] and faster RCNN [190] have been developed to avoid these problems. While GPU-based embedded platforms are preferred to shorten training and inference time, high energy consumption restricts the use of these platforms. High energy consumption does not fit the optimal energy profile, so research has focused on ASICs or FPGAs being as fast as the GPU and requiring less energy.

Study [191] provides an excellent example of YOLO-v3 and SVM/Random Forest classification. The study gave YOLO-v3 and faster RCNN performance comparisons and discussed their similarities and advantages. The study involves a pedestrian who intends to cross the street to warn the driver in case of risk. Here, they defined the process of crossing as situations such as the pedestrian standing position and the direction of rotation by creating the CNN model with the depth detection camera. In the first case, a 2D image is produced by considering the specific regions of the pedestrian; then, the depth information points make it possible to obtain this image in 3D. The CNN model has been trained end-to-end with data sets of pedestrians in different postures and scenes. The upper part of the body is the shoulders, neck, and face, which are the ROI to describe the pedestrian situation. The study used an Intel Core i7-7700HQ CPU 2.8GHZ 16GB RAM, NVIDIA GeForce GTX1060, and obtained a 94% accuracy rate with a data set with 20000 images of right shoulder, left shoulder, neck, and face taken with a ZED camera. The authors set epoch and batch sizes to 25 and 128 for the network training data, respectively.

In another article using yolov3-tiny (lite model of yolov3) and an independent logistic classifier, the authors in [192] reported that they employed the transfer learning approach for pedestrian detection in the Caltech US data set with a sensitivity of 76.7% and an inference time of 202 FPS. Using the COCO data sets, they stated that Yolov3 is the best CNN model with 57.9% sensitivity and 20 FPS inference, and Yolov 3-tiny works with 33.1% sensitivity and 202 FPS inference. Yolov3-tiny is suitable for use in embedded platforms, with high-speed processing and low memory requirements while offering low

accuracy. They stated that they reached an accuracy rate of 76.4% and a processing speed of 202 FPS with the method they presented in their study using the Caltech US data set. Original yolov3-tiny provided a 68.9% accuracy rate at 220 FPS processing speed for similar conditions. The authors conducted their experimental study on a Linux-based computer with an i7 CPU processor and NVIDIA 960 GPU. They created the CNN model with the DarkNet DL framework using the NVIDIA software accelerator CUDA 10 and cuDNN7 and implemented the original Yolov3 application on the NVIDIA TITANx GPU. The authors also confirmed that their proposed method reached 98% accuracy with different data sets that did not belong to the Caltech US data sets.

With the development of DL algorithms, researchers obtained excellent results in pedestrian detection in average weather conditions. Studies continue without slowing down to detect vehicles and pedestrians in unfavorable weather conditions. Tumas et al [193]. They stated that they reached an average sensitivity of 89.1% with Yolov3 DLL in their research using the ZUT (16-bit thermal data-sets) data set. They used state-of-the-art network training methods such as Yolo DNN Yolov3 and Tinyv3 and images with a depth of 16 bits and a resolution of 640x480 instead of 8 bits to increase Yolo's accuracy. 3k iteration, while they obtained only 66.3mAP for Tinyv3. When they raised the image quality to 640x480, they achieved a 6% performance increase for Yolov3, while they achieved the same performance for Tinyv3 for 416x416. In addition, they stated that while the low-pass filter they added to the system achieved a 4.3% improvement for Yolov3, this rate was 4.1% for Tinyv.

In the article related to existing pedestrian detection algorithm development by R.M. Castelino et al. [194], the authors presented a framework that combines SVM and eXtreme Gradian Boost (XGBoost) learning models with HOG feature extraction frameworks and improves the classification performance. While the initial state-of-art of the PSU and INRIA data sets they used for the accuracy of their work was 55% and 54%, the state-of-art of the PSU and INRA data sets with improved performance was 86% and 82%, respectively. The authors used an Intel Core i5-7300HQ CPU, 2.5GHz, 8GB RAM, and NVIDIA GeForce GTX 1050 GPU for their experiments. They applied the framework to the PSU data sets to detect pedestrians and then validated it in the INRA data sets.

The study involving the prediction of moving objects using DL technology for A/DAS

was carried out by Tsai et al. [195] on a three-dimensional convolutional platform with 20FPS performance with 92.8% pedestrian crossing accuracy estimation and 94.3% vehicle blocking behavior detection on a GPU based embedded platform. In the system suggested by the authors, accuracy increases with pre-training 3D convolution. The authors also stated that their system might be used in embedded platforms like NVIDIA Jetson Xavier due to its low complexity and memory space. All experimental studies were processed on the server with NVIDIA GTX 1080 Ti GPU, and they observed that they reached 20FPS on the embedded platform.

Using FPGA-based embedded platforms in such applications is inevitable when considering energy efficiency and performance. In [196], the authors have compared the performance of NN packages such as YOLO, GoogleNet, and AlexNet used in computer vision object detection and classification over GPU and FPGA-based architectures in recent years. The authors stated that NVIDIA Tegra X2 and Xilinx Ultra-scale get similar results in similar parameters in complex tasks such as detection. Here, they underlined that they used the FPGA platform without being optimized. However, it is vital to note that similar FPGA platforms show significant differences in different NN packages. The reason is that high-level synthesis (HLS) tools for using CNNs in FPGA platforms still need to be improved. The classification task stated that GPU-based platforms are superior to FPGAs with FPS rates. GPUs reached a speed of about 600 FPS, while FPGAs remained within the 60 FPS limit. Regarding energy efficiency and flexibility [197], FPGAs are more advantageous than others in similar applications. As we observed from the published works, designers can smoothly utilize recently developed IDE (Xilinx Vitis) solutions in AI, ML, and DL algorithms of FPGA-based ultra-scale platforms.

It is only possible to mention some of the work done in this field here. For this reason, we put the current studies regarding pedestrian and vehicle detection in recent years into a table. As in other studies, there are various possible methods or approaches based on hardware, software, framework, and library in studies related to pedestrian and vehicle identification. Although we cannot precisely compare them, we consider a comparative approach between price and performance. Table 3.7 and Table 3.8 compare the studies conducted in this field.

Table 3.7: A/DAS vehicle & pedestrian detection tasks implementation information in the literature

Ref.	Mission	Classification	data-sets	Embedding Methods	Accuracy (%) or mAP
[191]	Pedestrian movement	Yolov3, faster R-CNN SVM/Random Forest	Own	2D space to 3D space	94
[192]	Vehicles & Pedestrian det.	Yolov3 Logistic classifier	COCO - Caltect US	Transfer learning approach	57.9 - 76.7
[193]	Pedestrian detect.	Yolov3 DNN based detec.	ZUT	16-bith depth modification	89.1
[194]	Pedestrian detect.	HoG, SVM, XGBoost	PSU - INRIA	hyper-tuning	82 - 86
[195]	Vehicles & Pedestrian det.	Binary classifier	UCF101	3D Convolution	92.8 - 94.3
[196]	Vehicle detect.	CNN - Faster R-CNN	ETH	Pruning	90
[197]	Pedestrian detect.	CNN,SVM	Cladect pedestrian	k-means Quantization & pruning DeepPed Pipeline	N/A
[198]	Pedestrian detect.	HoG future based cascade classifier	black-box include 320 images	Binary tree classifier technique	93
[199]	Pedestrian & Vehicle detect.	soft-cascade AdaBoost classifier K-means clustering	KITTI - Caltech	Feature extraction	82.9
[200]	Object detect.	Improved YOLOv2 Filtering & Feature extracts	DOTA	Network quantization Layer fusion Unified imp. of multi. convolution	0.6732mAP
[201]	Pedestrian Ident.	Adaptive thresholding Features extracts	N/A	Data acquisition Real-time process	N/A
[202]	Vehicle detect.	RBF classifier Gray image Filtering, Edge detect. Morphological op.	Own	ROI Adopting RBF Network	97
[203]	Pedestrian detect.	SVM	Caltech	Hardware map.	N/A
[204]	Pedestrian detect.	SVM	Chrysler	N/A	94.04
[205]	Pedestrian detect.	Regression	Caltech	Lightweight conv. blocks & Model compression techniques Pruning & quantization tech.	92 mAP-87 Recall 90.54
[206]	Object behavior prediction	Yolo-v3 CNN based classification	UCF101	3D Convolution Feature extraction Thresholding	Rec. 92.8 Det. 94.3 Emerge 95
[207]	Pedestrian detect.	HoG, Multi-stage SVM	INRIA-MIT	Hardware/software co-design	Recall 94 Precision 92
[208]	Pedestrian detect.	Multi-scale region covariance descriptor (MSRCD), SVM	INRIA	Local Binary Patterns	18.94 speed-up 48.1 memory gain

3.2.4 Driver's Behaviours

Another essential part of A/DAS systems includes studies to reduce or eliminate accidents with the risk of death and injury resulting from driver behavior. According to World Health Organization data, approximately 1.35 million people die yearly from traffic accidents. Distracted drivers cause the vast majority of these deaths. Although the accidents vary by age and gender, drivers' lack of attention is often driven by texting, taking photos, self-care, fatigue, and an unfavorable driving environment. We can divide these into distraction (visual and cognitive) and fatigue. It is crucial to examine each category and process the types of information related to them. With the advancing technology in recent years, the functionality of A/DAS systems has increased, and as in every field, A/DAS

Table 3.8: A/DAS vehicle and pedestrian detection tasks implementation information in the literature

Ref.	Library, Interface, Language	Hardware Platform	Image Size	FPS	Cpp ($\times 10^{-5}$ \$)
[191]	Keras	Intel Core i7-770HQ CPU 2.8GHZ 16GB RAM GeForce GTX 1060	75x75	45	592.59
[198]	MATLAB	Quad Core 3.4 GHz, 32 GB, GPU 1080ti	1920x1080	2.5	77.16
[193]	Baumer GAPI SDK OpenCV	i7 8th Gen. NVIDIA RTX2070 GPU	640x480	N/A	N/A
[194]	OpenCV	i5-7300HG CPU 2.5GHz 8GB RAM NVIDIA GeForce GTX 1050	N/A	N/A	N/A
[205]	N/A	Intel i7 CPU, NVIDIA GTX960 4GB 1024 cores	224x224	35	75.9
[199]	N/A	Intel(R) Xeon(R) E5-2620 v4 CPU 32 GB Memory	1242x375	2.41	N/A
[202]	MATLAB	N/A	250x190	N/A	N/A
[192]	N/A	NVIDIA Titan GPU, NVIDIA GTX 960	416x416	220-202	2.62 - 57.21
[197]	N/A	NVIDIA Jetson TK1	640x480	10	6.51
[204]	Caffe	NVIDIA Jetson TX1	720x480	5	41.78
[206]	N/A	NVIDIA GTX1080ti, NVIDIA Jetson AGX Xavier	1920x1080	10	10.12
[195]	N/A	NVIDIA Jetson AGX Xavier	224x224	20	69.75
[196]	PipeCNN - XFCDNN	Xilinx Ultrascale - NVIDIA Tegra X2	408x408	60- 600	14.011-0.8009
[207]	OpenCV, C++, Vivado	Xilinx Zynq FPGA ZedBoard	640x480	42	3.87
[203]	Vivado	Xilinx ZC 702 FPGA	1270x720	60	1.7989
[200]	PyTorch and TorchVision Vivado VHDL, C & Python	Xilinx ZYNQ xc7z035 NVIDIA Titan Xp GPUs	1024x1024	N/A	N/A
[201]	OpenCV Python based testbed	DCA1000 FPGA	N/A	25	N/A
[208]	N/A	Xilinx ZYNQ SoC XC7Z020	128x64	25	14.6

reduced the negativities caused by examining driver behaviors. In the A/DAS systems, designers use biological signals, cameras, and warning sensors to eliminate or minimize behavioral inattention. It is possible to stimulate the correct driver behavior with emotional state analysis with EEG, ECG, and EMG. Another way of relieving driver inattention can be a design that allows the driver to perceive the driver's gaze, behavior, and actions like signs of eye fatigue with cameras and act accordingly. Although in-car lighting or the driver's wearing accessories such as sunglasses can make the situation harder, depth or infrared cameras might overcome this issue. In addition, another method presented by Xu et al. in [209] benefited from the Doppler effect in their study to improve driving safety. Using smartphone audio devices, they created the Doppler profile to recognize careless driving events such as turning around, eating, and collecting a falling object. Thanks to this inference, they developed an early recognition system (ER) that warns by determining driver behavior and what to do in response using ML algorithms.

This part of the work presents state-of-the-art studies preventing accidents caused by driver inattention. As it is known, the studies carried out in A/DAS systems are hot topic studies, and hence, researchers have published numerous articles, statements, and presentations concerning the subject so far.

In A/DAS systems, the driver's vehicle control and energy efficiency are also carefully examined. An advanced A/DAS system must undoubtedly be energy efficient. Designers can use ML-based techniques to increase the efficiency of the vehicle against these behaviors by classifying it according to the spiritual model of the driver (aggressive, average, and calm). In the study [210], the authors addressed these three mood changes and stated that an ANN-based efficiency-oriented model showed 90% general accuracy compared to other driving scenarios. The training data was obtained based on a driver's behavior while traveling at 90 km/h in a simulation environment. While getting learning data, they considered the characteristics of longitudinal acceleration, pedal angle, and vehicle speed. The driver communicates directly with the car and samples the data at intervals of 100 ms. They obtained an average accuracy of 92.7% for training and inference in the study using an ANN-based feature extractor and classifier. The authors should have provided information about the platforms they employed, encompassing hardware, software, frameworks, etc. Another study [211] investigated the longitudinal, lateral, and braking behaviors of 32 drivers with the Mobileye 630 type vehicle that they used twice on the same track. The authors obtained driver behavior, environmental information, and vehicle dynamics with controller area network (CAN) bus and video cameras. In line with the obtained results, we can see that the researchers examined speed, longitudinal plus acceleration, longitudinal minus acceleration, deceleration, and progress in actions such as accelerator and pedal. We observed that A/DAS improved only in longitudinal minus acceleration (LMA) and longitudinal deceleration and time headway (THW). Drivers used urban roads, urban expressways, and highway roads for the trials. The authors also noted that driving safety was significantly improved when drivers were exposed to A/DAS because lane departure on the city's urban expressway and the freeway were reduced, but not on the urban road. They stated that this was the inconsistent lane detection on urban streets. When we consider the relative speed and braking events while A/DAS is active, THW times are reduced for each route in all tests, which shows that A/DAS ensures traffic safety by reducing close collisions. The path length used for the experiment is 45km for the freeway, 50km for the urban expressway, and 12km for the urban road, 107km in total, and the average time is 126 minutes.

Forward vehicle collision warning (FCW) is an essential part of A/DAS. It is necessary to

detect the vehicle ahead and accurately locate and extract distance to avoid a collision. The study [212] mentioned the camera system that combines absolute distance measurement and vehicle detection for FCW. Unlike the driver's acceleration, braking, and stopping, it is a method based on the inference of the situation from abnormal facial expressions with the in-car camera system. There are 17,500 sample images and six facial expressions, such as right-left gaze, drowsiness, yawning, and phone calls. They created their own data sets to verify abnormal driver behavior detection. The authors stated that camera calibration is required only for distance measurement. It can process at 30 FPS with the KITTI data sets to validate multi-scale vehicle detection. They presented that their systems quickly met the real-time requirements with 25FPS, with three times faster processing time than the work shown in [213].

Sensor fusion-based, or ML-based driving simulators are generally used in the literature to identify and define driver behavior. Drivers' reactions are measured thanks to the sensors in the driving simulator. These measurements allow us to analyze driver behavior. Thus, we can obtain data sets from different traffic conditions, weather conditions, and human-like emotions of the driver. A real-time method (adaptive, NNs, fuzzy logic, etc.) is obtained with these data sets for autonomous or A/DAS systems. This method helps the driver in traffic situations.

Researchers have also been using FPGAs in real-time and giving outstanding results. The work presented in [214] illustrates FPGA-based driver behaviors' prediction, like high-speed gaze detection for A/DAS. The authors obtained better results than methods in similar applications by combining the enhanced semantics-based vague image representation (ESVIR) approach and fully connected NNs. Unlike SVM, CNN, and Adaboost, the ESVIR algorithm uses basic patterns during pixel scanning. Since the algorithm minimizes the size of the NNs as a 1D-codes basic pattern, it provides very high energy efficiency thanks to low memory requirements and complex processing load. Their study using the Quartus II FPGA model stated that there was only 409.26mW of power consumption for the total throughput at an 80FPS frame rate with a 640x480 pixel image. The Adreno 530 GPU-based architecture for a similar workload has 16 FPS, and the i7-3770 CPU-based architecture has 20 FPS at a 227x227 pixel image. When we consider the experimental results, regardless of the applications in different climatic and daylight

conditions, the study seems successful for video-based eye tracking.

Since it is impossible to examine and explain all the studies in detail in the literature, we have tabulated some of them in recent years in this section. Considering the current dynamics, the methods, hardware, and libraries used in the studies are different, so it is not straightforward to make a fair comparison. However, close comparisons of similar studies are given in Table 3.9 and Table 3.10.

3.2.5 Including More Than one A/DAS Task

In A/DAS, tasks are handled individually, such as traffic sign recognition, lane or pedestrian detection, and applications where all tasks are combined. Multiple tasks, such as facial recognition, word recognition, detection, parking, etc., are examples of combined A/DAS.

In parallel with the developing technology and increasing needs, we are sliding towards smart cities when discussing smart grids, phones, and home automation systems. The growing and expanding city life brings problems with increasing the human population. One of these problems is the need for more parking for the increasing number of vehicles with the growing population. Unfortunately, intelligent parking solutions, one of the application parts of A/DAS, cannot meet the rising demand. Moreover, the scarcity of research and studies in this direction reveals this. Intelligent parking solutions shorten the time needed to search for a parking space while reducing urban traffic congestion, providing a more sensitive environment for the driver and the environment. For intelligent parking solutions to be viable, human-machine interaction (HMI) and vehicle environment interaction must be constantly interconnected via a network connection. There are studies in this area, albeit limited. Google Maps or Yandex maps are needed with a server to ensure that the vehicle and previously mapped areas (parking and transportation information) communicate.

In [230], the authors stated that they designed the adaptive parking system at a more affordable cost, thanks to the advanced DL algorithm and a camera network. They used a particular camera system with a zoom lens and a motor where the vehicles determine their entrance and exit to the parking area with their license plate numbers. The authors' work consists of a network structure for ground and top view cameras, end devices where

Table 3.9: A/DAS driver's behaviour detection tasks and features in the literature

Ref.	Mission	Classification	data-sets	Embedding Methods	Processing data	Accuracy or Precision (%)
[210]	Driving efficiency	ANN	N/A	Feature selections NN classifier	Steering wheel Brake Gear shift	94.5
[211]	Driving performance & braking behavior	Feature extract	N/A	Road type calibrat. Traffic state calib.	Camera & Brake LIDAR & CAN	N/A
[212]	Adaptive FCW	Feature extract	Own & KITTI	Compute channel feature & apply boosted tree & multi-scale sliding window	Camera, Image	Det. 96.61 mAP 91.6
[215]	Driver behavior	Feature extract LSTM	Own (Self-collected)	Unsupervised spectral clustering	Video frame GPS & ECU inf.	80
[214]	Video-based eye-tracking	Fully-connect.NN Morphological Operation Thresholding	YALE & Senthilb & Own	Semantics-based vague image representation	Camera, RGB	92
[216]	Driver Behavior Characteristics Ident.	ANN	Own	Genetic algorithm Particle Swarm Op. Back propagation NN	Camera, Radar & GPS	N/A
[217]	Driver Behavior Analysis	N/A	Own	Modeling a driver in a vehicle PERCLOS	Accelerometer Gyroscope Microphone GPS, Camera	N/A
[218]	Driving behaviors prediction	Thresholding Fuzzy approach	N/A	Hidden Markov Model Combined by Fuzzy Logic	Driving simulator	86
[179]	Head pose estimation	CNN & RNN	ICT-3DHP & Biwi	N/A	RGB image & Depth	N/A
[219]	Driver state	CNN	Own	Compression of heavy baseline model	RGB images	91.2 - 89.5
[220]	Driver state	CNN	DROZY,Toucango	Classify images	RGB image	89
[221]	Driver behavior	CNN	Own	Computer vision technique Recurrence plot technique	Acceleration Gravity, Speed Throttle, RPM	99.98
[222]	Driver behavior	k-NN,PCA,HCA	Own	Nonlinear auto regressive NN	WheelCollider Brake, Speed	Reduce 20% speed error 7.2% energy consumption
[223]	N/A	Fusion-based CNN LSTM	Honda Res.(HDD)	Goal-oriented action	CAN,GPS,Gyros LIDAR, Camera	32.71 mAP
[224]	Effective drive state	K-NN Random forest	Own	ingular Spectrum Analysis	EEG, Camera	86.7
[225]	Driver Behavior	fuzzy c-means clustering, PSO	N/A	BP neural network	Throttle, Brake SENSO-wheel	85
[226]	Fatigue state of the Driver	Adaboost learning Algorithm Morphological Op.	Own database	HW/SW Co-design	RGB image, Binary image	N/A
[227]	Drivers' visual -distracted take-over	Adaptive adjustment of time budget	Own	Multiple regression model & Face orientation	Acceleration Deceleration brake, angle Camera	86.3
[228]	Gaze tracking	Fusion based classifier	GTEA-plus OSdataset	Model-based estimation	3-D eye model Camera Point of Regard	96.37 81.84

transactions are made, a cloud service database, and a web application. They used Raspberry Pi and NVIDIA Jetson TX2 as edge devices and an RPI interface for ground and top-view cameras. They run the Tiny Yolo model on the Jetson Tx2 to control the vehicle's entrance, exit, and stay time in the parking lot. While running OpenALPR on RPi

Table 3.10: A/DAS driver's behaviour detection task implementation information in the literature

Ref.	Library, Interface, Language	Hardware platform	Image size	FPS
[210]	MATLAB	PC & Logitech G29	N/A	N/A
[211]	N/A	OXTS RT2500INS, CANoe, Ibeo LUX−4 LIDAR	N/A	THW 2, 2.5, 3.33
[218]	N/A	SCANeR	N/A	N/A
[216]	ICC & MATLAB/Simulink	PanoSim RT, dSPACEDS1006	N/A	N/A
[222]	Human-machine interface	Unity game engine-based driving simulator platform Intel Core i7− 7700K@4.20 GHz, RAM 64.0 GB Logitech G27 Racing Wheel	N/A	N/A
[224]	PreScan	Force Dynamics 401cr driving Sim.	640x480	N/A
[225]	MATLAB/Simulink, CarSim	dSPACE DS1006 real-time system sim.	N/A	20
[228]	Emgu CV, Kinectface, Biometric API	Microsoft Kinect v2.0	N/A	30
[229]	Virtual driving environment	Dynamic motion driving simulator	N/A	N/A
[227]	N/A	High−fidelity driving simulator & FPGA	1080P	N/A
[215]	TensorFlow, MATLAB	Cloud platform Vehicle−mounted data collection system Wireless transmission system On−Board Diagnostics (OBD)	N/A	0.33 to 0.2
[217]	OpenCV and DLib	Android Smartphone	N/A	9
[212]	Caffe	NVIDIA TX1 with 56 Core Pascal GPU	1280x720	25
[179]	Keras, C++, Microsoft Kinect SDK	NVIDIA Jetson TX1 - NVIDIA Jetson TK1	640x480	30 – 19
[219]	Caffe & DL framework	GTX1080 Cuda 2560 Cores, NVIDIA Jetson TK1	640x480	90.1 - 14.9
[220]	OpenCV Haar Cascades, TensorFlow Lite	Inte Core i7 3.1GHz, NVIDIA GeForce 920M Qualcomm Snapdragon 430 MSM8937	224x224	5 to 10
[221]	TensorFlow, Keras, Python Pyts	CPU i7 32GB RAM, NVIDIA GeForce 1080Ti	150x50	N/A
[223]	N/A	Intel i5 6600K 3.5 GHz Quad Core	1280 x 720	30
[226]	Vivado HLS, C++, OpenCV	Xilinx Zynq ZC702 FPGA	750x450	N/A
[214]	Quartus II, HDL	Altera DE2 115 FPGA Cyclone IV E	640x480	80

for plate recognition, they provided feature extraction of vehicles using OpenCV. Using PKLot [231] and CNRPark [232] data sets, they reached 99.51% accuracy with a total of 7.11ms for inference.

In another A/DAS parking application, the authors [233] implemented the Parking Line Departure Warning System (PL-DWS) on the NVIDIA Jetson Tx2 embedded board using the VGG-16 [142] CNN model. The difference from [230] of this study is that the authors aimed to park the vehicle in the entire parking area by detecting the parking lines with the four-camera system placed on the car. They used three different platforms to test system behavior and compare accuracy rates. They realized a 224x224 pixel image with NVIDIA Jetson-Tx2 embedded board at six FPS with 98.01% precision and 99.13% recall. The authors have run the Keras 2.1 [234] and Tensorflow 1.5 [235] DL frameworks using Python packages. This study observed no hardware or software optimization to increase the FPS rate. For a 224x224 image data set, 6FPS is lower. The FPS value can be improved by using optimization methods or different CNN models.

One of the essential tasks of A/DAS is classification using the computer vision technique.

The processing time varies according to the quality of the unprocessed image obtained with the camera or other devices. This process makes it impossible to process in real-time systems due to continuous data flow and high-resolution photos. In the study of [236], the authors have designed an algorithm that enables the automatic filtering of video stream images by utilizing their properties to be used in any A/DAS object detection and classification applications. They assigned the criteria corresponding to the property values of the image and the evaluation function corresponding to each criterion to evaluate the quality of the image obtained. The image is converted to 256 grey-scale modes using the colorimetric conversion technique while preserving the brightness of the input video stream. The authors preprocessed the images on the NVIDIA Jetson TX2 via the Intel I350-T2 network card, and the HD images were obtained from the two cameras.

In A/DAS implementations, most algorithms are often inefficient and costly to use for a task. Performing most functions on a single algorithm and a single development board will increase efficiency and reduce costs. [113], the authors stated that they designed an A/DAS application using a single development board for multiple tasks such as semantic segmentation, lane detection, object detection, and orientation estimation. The authors proposed a lighter semantic segmentation network due to the Feature Pyramid Network (FPN) structure. Unlike standard semantic segmentation, they can achieve higher accuracy for real-time applications and reduce workload. In typical methods, the algorithm usually requires several TFLOPs or hundreds of GFLOPs for an image, while it cannot guarantee the frame rate in real-time applications. In addition, we observed that they increased the program's efficiency from 76.9% to 97.7% with the optimization improvements they made in hardware and software. They used the Xilinx ZU9 development board as hardware and reached 55 FPS in real-time.

Similarly, in the [138] articles that offer multitasking for A/DAS, the authors presented the multitasking semantic attention network (MTSAN) to deal with semantic segmentation and object detection tasks in real-time applications. The authors also implemented a Semantic Attention Module (SAM) that extracts semantic cues from the segmentation subnet independent of other application parameters, significantly reducing the misinformation rate and improving performance similarly. The specified structure comprises the backbone encoder, perception and segmentation subnets, and the semantic attention module. The

backbone encoder is used for image processing and extraction of important information. JacintoNet [237], suitable for lightweight and embedded devices with open source code, was preferred instead of very deep architectures (AlexNet et al.). The authors compared their work using multiple CNN models and data sets. The experimental results achieved 10FPS for 512x256 image size with NVIDIA Jetson Xavier hardware and 15FPS for the exact image size with Texas Instrument TDA2x hardware.

In A/DAS, handling the tasks as a whole (multitasking) rather than separately and creating network models increases communication and harmony between the functions. Otherwise, each task requires training, inference, and equipment. In light of this information, the study performed in [238] presents an algorithm that includes a multitasking learning (MTL) network and the joint training of these networks. In their work, the authors mentioned the prediction of contextual information (identification of information between hidden field and input), which may be considered the first in MTL. This method uses neural mutual information calculation to compute specified contextual information. This process consists of formulation, mutual information restriction, and lower limit restriction steps. In the study conducted by the authors for traffic scene perception, they stated that the experimental results improved state-of-art in both single and multitasking environments compared to similar studies. Their work used NVIDIA Titan X GPU and i7-6700K CPU, achieving 25FPS in a 1280x720 video image.

So far, we have briefly explained the studies of multitasking A/DAS applications used for different purposes on different platforms. After that, as in other A/DAS assignments, we presented the work done as A/DAS multitasking in recent years to the reader in Tables 3.11 and 3.12.

3.3 A/DAS Application Challenges

After examining the utilization of A/DAS applications, I briefly explained the difficulties, limitations, and concerns encountered during the implementation in this section. I discussed how these affect the ML models in terms of performance and accuracy.

Table 3.11: A/DAS multi-tasks and associated features in the literature

Ref.	Mission	Classification	Data-sets	Embedding Methods	Processing data	Accuracy (%) or mAP
[230]	Smart parking	CNN	PKLot CNRPark	Edge computing Avanced DL algorithm Data analytics	Camera, LIDAR RGB image	99.51
[233]	Parking Line Departure Warning	DL classifier	Own	Image processing and Labelling	AWM Camera	Prec. 98.01 Recall 99.13
[236]	Automatic video stream images evaluation	N/A	N/A	Median Filter Image preprocessing	Camera	N/A
[113]	Multi-task learning framework	CNN	Cityscape	SW & HW Optimiz. Light-weight SSD	RGB Image	97.7
[138]	Multi-task Semantic attention network	RCNN, SVM	Cityscape COCO	Post-processing method SSD detector	Camera Image	98.31
[238]	Traffic Scene Recog.	CNN	HSD	Multi-task learning network	Camera image	70
[232]	Decentralized parking lot occupancy detection	RBF kernel SVM	Own, PKLot CNRPark-EXT	Deep CNN (mini-AlexNet)	Smart Camera RGB image	98.27
[239]	Parking space recognition	Thresholding	N/A	Panoramic vision Distortion correction Perspective transform.	Camera image	97.63 79.77
[135]	Multi-Task Network Pruning and Embedded Optimization	CNN, Pixel-level clasifier Quantization	Woodscape	Camera-based Deep Learning algorithms	Camera image	Det.68.17 mAP
[240]	Automatic Parking	Global Nearest Neighbor Filtering	Own	Matching and Fusion	GPS, Camera Radar, LIDAR	N/A
[241]	Collaborative mapping in parking systems	Binary	OpenStreetMap bird's eye view map	Trapezoidal cell decomposition & Voroni diagram based	GPS, IoT CCTV LIDAR	N/A
[242]	Parking assist system	Haar Cascade Classifier AdaBoost	N/A	Deep learning, Simultaneous localizing & Mapping	Camera image	97.52
[243]	Automated parking systems	SVM	Own	Computer vision algorithms	Camera, LIDAR	N/A
[244]	Accelerating Image Processing Functions in A/DAS App	Canny edge det. Feature extract. Edge detect Filtering	N/A	Heterogeneous processing pipeline & Image processing	24-bit RGB	N/A
[139]	Semantic seg. Multi-object det. Drivable area seg. Line det.	CNN Feature extract. Edge detect	Waymo, BDD100K CityScape KITTI	Heterogeneous comp. Multi-threading pipeline Multi-task learning	Camera	57.95 mIoU 51.29 mAP 82.63 mIoU 43.65 IoU
[140]	Object detection Semantic segmentation, and Disparity estimation	CNN	Cityscapes	Depth-wise Point-wise convolution Bilinear interpolation	RT Camera Image	41.1 Perf. Improve
[245]	Semantic segmentation and Object detection	YOLOv4, CNN	Cityscapes	Encoder-decoder CNN	RT Camera Image	90

3.3.1 Power Consumption

Reducing power consumption is a critical goal for any battery-powered application and represents one of the principal constraints. Power consumption is closely linked to the NN architecture, image size, resolution, and the hardware used in A/DAS. Understanding the specific application requirements provides insight into the necessary processing performance. This understanding can be challenging in A/DAS, where ML is extensively employed and performance is highly application-specific. For applications using CNNs, various parameters need consideration, including the number of layers, network depth, image resolution (if classifying videos), frame rate requirements, and bit-width. It is

Table 3.12: Implementation information for A/DAS multi-task applications

Ref	Library, Framework Language	Hardware Platform	Image Size	FPS	Cpp ($\times 10^{-5}$ \$)
[230]	Tensorflow, Tiny YOLO DarkNet	Raspberry Pi, Jetson Tx2	1920x1080	14.06	6.24
[140]	PyTorch	Personal Computer	1024x512	30	N/A
[232]	OpenCV, Caffe	Raspberry Pi 2 ARM Cortex A7 CPU, 1GB RAM (Camera modules)	2592 ×1944	N/A	N/A
[240]	Python	Mobileye 630, ARS408 Radar, LIDAR	N/A	N/A	N/A
[243]	OpenLVDS, OpenCL, OpenGL	Electronic Control Unit and Camera housing	2560x1440	30-60	3.10
[241]	OpenStreetMap	LIDAR, IoT based systems	N/A	N/A	N/A
[242]	OpenCV	4-Cores 3.5 GHz	800x600	10	10.41
[233]	Keras, Tensorflow, Python	GTX 1050 GTX-1080Ti NVIDIA Jetson TX2	224x224	20 80 6	11.9 36.87 27.2
[236]	OpenCV, VisionWorks	NVIDIA Jetson TX2, Network Card Intel I350-T2	720P	5.95	0.645
[245]	YOLOv4, OpenCV	Nvidia Jetson TX2	416x416	N/A	244.2
[138]	Linux NVIDIA CUDA	NVIDIA Jetson Xavier Texas Instruments TDA2x	512x256 512x256	10 15	30.5 20.3
[238]	PyTorch	NVIDIA Titan X GPU Intel Core i7-6700K CPU	1280x720	25	4.77
[239]	OpenCV, C++	NVIDIA GeForce GTX1060 - Intel Core i5-9400F	1024x768	50, 5	25.4, 95.4
[244]	Vivado HLS tool Python OpenCV	PINQ-Z1 Zynq-7000 ARM/FPGA SoC	1920x1080	60	0.160
[113]	DL Development Toolkit	Xilinx ZU9 FPGA	512x256 480x360	55	15.7
[135]	N/A	Prototype platform a low power SoC	800x600	10	4.16
[139]	Vivado HLS tool Python OpenCV Pytorch	Kria KV260 Vision	1920x1080	25	0.48

important to note that selecting high-performance hardware entails trade-offs regarding cost, physical size, and power consumption. Rather than opting for networks designed for high accuracies, such as ResNet, AlexNet, and GoogleNet, which are suitable for embedded architectures but come with high computational demands, it might be preferable to use networks that either compromise on accuracy or reduce the computational load by implementing quantization strategies within the network structures [139, 246, 247].

Additional factors influencing power consumption include the long wake times of devices, the complexity of code blocks, the type of device memory, and the absence of optimization [21, 246]. The chosen CNN models must be deployed on accelerator platforms due to their substantial floating-point computation requirements. Parallelized computing offers a solution to traditional serial computing by enhancing speed and power efficiency. Hybrid configurations that use CPUs with GPUs in Edge AI computing provide some acceleration advantages.

However, the substantial energy consumption of GPUs often limits their use, especially in compact embedded systems where power efficiency is paramount. FPGAs are particularly suitable for applications requiring low power consumption due to their ability to perform

highly efficient and low-latency operations in edge AI contexts. The inherent capability of FPGAs for parallel programming further distinguishes these devices [248–250]. Another hardware option for AI and ML applications is ASICs, recognized as highly energy-efficient hardware accelerators [251]. However, their need for reconfigurability makes them unsuitable for applications that require frequent updates or modifications.
The computational costs in NNs, primarily driven by matrix-vector operations, can be quantified by the number of Multiply-Accumulate (MAC) operations:

$$\text{MAC ops} = N_f \times F_s \times I_s \tag{8}$$

where N_f is the number of filters, F_s is the filter size, and I_s is the input feature map size. These operations are directly correlated with power consumption, with increased complexity leading to higher power needs. Techniques such as quantization, which reduce the computational load and power requirements by lowering the precision of calculations, allow for more efficient operations:

$$\text{Quantized MAC ops} = \text{IntOps} \times \text{RBW} \tag{9}$$

where MAC ops is the Multiply-Accumulate operations, IntOps is the Integer Operations, RBW is the Reduced Bit-width.
Quantization effectively reduces the bit-width required for each operation, thereby decreasing the energy per operation and enhancing system power efficiency.

3.3.2 Execution Time

Respecting execution time constraints in critical applications such as A/DAS is not just a technical requirement but a crucial aspect that directly impacts the performance of these applications. The execution time, which is the time needed to run inference and measured in Frame Per Second or FPS, is a key metric. Due to the different clock frequencies of hardware accelerators, we cannot rely on clock frequency alone to determine the time for some algorithms to complete their tasks. The nature of the architectures can lead to low-clock-frequency processors outperforming higher-clocked processors in certain tasks. Therefore, it is imperative to run the desired task in as few machine cycles as possible

to reduce the execution times of algorithms, thereby enhancing the application's overall performance.

The execution times of arithmetic operations frequently used in ML applications can vary significantly. For instance, no multiplication or division can be made in an additional operation time. This is why it is often intractable to operate on real numbers directly in digital processors such as FPGA, and designers must transform them into different data types with certain transformations, such as floating-point arithmetic or fixed-point arithmetic. These conversions, along with other factors, can significantly affect execution time. To ensure that the code block can perform its task within the specified time, it is crucial to optimize the code beforehand and make it suitable for hardware accelerators. Architectures with inherently parallel processing capability can complete the given task in a short time, even if the clock frequencies are low. Therefore, the choice of application-specific software or hardware architectures for performance is a critical decision that can directly impact the execution time of the application [86, 252, 253].

3.3.3 Accuracy

Accuracy is one of the primary metrics used to evaluate the performance of classification models. It is defined as the ratio of correctly predicted observations (both true positives and true negatives) to the total observations in the dataset,

$$\begin{aligned} \text{Accuracy} &= \frac{\text{Number of Correct Predictions}}{\text{Total Number of Predictions}} \\ &= \frac{TP + TN}{TP + TN + FP + FN} \end{aligned} \tag{10}$$

As given in (10), accuracy is straightforward but can sometimes provide a partial view of model performance, particularly in cases of class imbalance. This metric assumes an equal cost of false positives and negatives, which often does not hold, especially in practical applications beyond the training phase. Other metrics, such as precision, recall, and the F1 score [254–256], are crucial to achieving a comprehensive evaluation. Precision, described in (11), assesses the accuracy of positive predictions.

$$\text{Precision} = \frac{TP}{TP + FP} \tag{11}$$

Recall, detailed in (12), evaluates the model's ability to identify all actual positive cases, which is critical in contexts where missing a positive instance can have severe consequences:

$$\text{Recall} = \frac{TP}{TP + FN} \tag{12}$$

The F1 score, described in (13), a harmonic mean of precision and recall, balances the consideration of false positives and false negatives.

$$\text{F1 Score} = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} \tag{13}$$

Moreover, the relationship between the amount of data and model accuracy is not straightforwardly linear, as more data does not invariably lead to higher accuracy. The benefit of additional data diminishes as the model approaches an inherent accuracy limit based on feature relevance and algorithmic learning capacity.

$$\text{Model Accuracy} = f(\text{Data Quantity}, \text{Data Quality}, \text{Algorithm Complexity}) \tag{14}$$

Equation 14 highlights that while increasing data quantity generally supports model training, improvements in accuracy are contingent upon the quality of the data and the complexity of the algorithm used. A robust evaluation of an ML model requires measuring accuracy and considering a suite of metrics to understand the model's performance fully. This approach ensures that ML models are developed accurately and aligned with their application's specific needs and constraints.

3.3.4 Memory

According to the research conducted by OpenAI in recent years, the data size used in AI/ML has increased approximately 300,000 times between 2012 and 2019 [257]. However, the size of AI/ML model data sets is also growing. The enormous models currently exceed 170 billion parameters [257] and are expected to increase much more. Thus, it is clear how big models will come out in AI/ML in the coming years and how big the memory requirements will be. For AI/ML inference to provide real-time responses, the memory must have high bandwidth and low latency [258]. Depending on requirements, AI/ML applications

choose one of three memory types: on-chip memory, high-bandwidth memory (HBM), and GDDR memory. While GDDR and HBM are high-performance external memory and are used and developed for the needs of the most demanding applications, on-chip memories [258, 259] are the fastest and most energy-efficient memories. However, their capacities are lower than those of GDDR and HBM. For this reason, on-chip memories are a good choice for systems with smaller NNs used for inference. HBMs are ideal for training, while designers prefer GDDR memories for inference and training in larger NN models and data sets.

Studies [260–262] have shown that classification can be done well in network models using one or two-bit quantization for weights and activation functions. The low precision arithmetic and small memory footprint offer an opportunity for expeditious and energy-efficient classification using FPGAs. The small memory footprint allows large-scale network parameters to be kept on-chip, thus eliminating the on-chip memory bottleneck. It shows us that FPGAs can have the potential to process up to Tera Operations Per Second (TOPs), especially in BNNs [263] architectures.

3.3.5 Scalability

CPUs are generally not used for large-scale ML because they are capable of sequential processing by nature, and data of such a large scale puts CPUs in a bottleneck. Unlike CPUs, GPUs allow fast data processing to parallelize (e.g., vector products) with the help of hundreds of embedded ALUs. Recently, GPUs have been extensively used in areas such as ML algorithm training and inference, data mining, and more, despite CPUs offering general-purpose use. [129, 130, 264–266]. Although GPUs are capable of high-speed computing, they negatively affect the use of these structures in terms of higher power consumption and the von Neumann (speed of the processors is much faster than the speed of the memory) bottleneck [267]. With their high efficiency, low energy consumption, and performance, ASICs are application-specific and thus not scalable, making them unsuitable for general usage. Unlike ASICs, FPGAs are acceptable for general purposes thanks to their reconfigurable logic blocks, programmable interconnections, and scalability [268]. Consequently, FPGAs are preferable in A/DAS applications rather than other hardware accelerator architectures.

3.3.6 Applicability and Flexibility

No matter how smoothly a model works with hardware accelerators, there is no optimum way to develop an exact model for each hardware. Hence, restrictions and adjustments are necessary for the model. It is also related to the suitability of hardware accelerators for software because not every hardware accelerator reacts identically to software due to its structure.

When comparing hardware accelerators, it is crucial to understand their unique features. FPGAs, for instance, stand out due to their re-programmable nature, which makes them a versatile option for general usage. Unlike ASICs, which are limited by their custom hardware structure, FPGAs can be adapted for each application, making them more flexible. In the case of GPUs, their software-defined data flow and complex memory hierarchy allow for efficient workload mapping. However, FPGAs flexibly surpass GPUs with their hardware-programmable DPS and BRAM blocks. This unique feature allows reconfiguration during operation, avoiding restrictions such as a fixed bus or SIMT. These highly flexible FPGA models have demonstrated exceptional potential in various ML applications thanks to their easy development platform, inherent parallel processing capability, low power consumption, and the flexibility they offer through their reconfigurable architecture.

3.3.7 Bottlenecks and Potential Solutions

In the rapidly evolving domain of ML applications within A/DAS, mitigating bottlenecks necessitates a multifaceted strategy encompassing technological innovation and methodological refinement. Key trends include Hw-Sw co-design, which optimizes algorithm compatibility with specific hardware platforms (GPUs, FPGAs, ASICs, CPUs), and model compression techniques (quantization, pruning, knowledge distillation) that streamline ML models for enhanced execution speed and reduced power consumption. The shift towards edge computing also facilitates real-time processing by minimizing latency through local data analysis. Developing specialized ML accelerators and energy-efficient hardware further underscores the industry's commitment to overcoming computational and power-related constraints. Adaptive and scalable architectures ensure system responsiveness

to varying computational demands, while collaborative and connected systems exploit V2X communications to distribute processing loads, thereby alleviating individual vehicle systems' burdens. Collectively, these approaches signify a comprehensive effort to address the integration challenges of ML in A/DAS, promising significant advancements in autonomous and semi-autonomous vehicle technologies.

3.4 Optimization Techniques for Hardware and Software

A designer needs optimization techniques to implement any ML model on platforms with limited resources. Thus, one can reduce the models' memory footprint without sacrificing usability. The optimization process should be good for model gains regarding accuracy and precision. Both metrics indicate how close a measurement is to the desired value. In the following subsections, we provide a brief overview of the optimization methods used in the literature.

3.4.1 Fixed and Floating Point Arithmetic

I delve into the crucial aspects of numerical representation in AI/ML algorithms, exploring floating-point numbers as the cornerstone for creating data representations. This foundational element necessitates the support of digital systems to execute floating-point calculations seamlessly. When a shortage of floating-point units in the CPU arises, designers may convert CPU operations from floating-point to fixed-point arithmetic, a process known as quantization. However, this transformation introduces challenges such as a loss of sensitivity, rounding errors, and additional time delays [269–271].

The need for complex and fractional calculations in AI and ML algorithms underscores the advantages of floating-point arithmetic over fixed-point alternatives. Despite its application precision, floating-point numbers' computational complexity and memory requirements can lead to a performance bottleneck. While floating-point arithmetic delivers accurate results, the inference latency tends to be high, posing challenges for ML/DL applications that demand both high accuracy and low latency. In response, designers actively explore alternative approaches, such as quantization and approximate computing, to mitigate the performance bottleneck associated with floating-point arithmetic. An il-

lustrative example of such an alternative approach is found in the work of Umuroglu et al. [260], who presented FINN, a fast and flexible FPGA accelerator framework based on a versatile heterogeneous flow architecture. Their study, positioned within the broader context of numerical representation challenges, showcases the efficiency of implementing BNNs as hardware accelerators integrated with an ARM CPU. They reported remarkable achievements, including the classification of 12.3 million images with 95.8% accuracy and an unprecedented 0.31us latency for the MNIST dataset on the Xilinx ZC706 embedded FPGA development platform [260]. This study is a testament to the viability of alternative methods, the BNN approach, in surpassing the limitations of floating-point arithmetic concerning inference accuracy and latency

3.4.2 Platforms

Designers use a variety of platforms in AI/ML algorithms for A/DAS, as shown in Table 1.2. Application-specific requirements frequently lead to platform differences. In other words, while speed is a critical parameter in one application, precision and accuracy may be more crucial in another one. Designers can apply platform optimizations in both the software and hardware partitions. In modern microprocessor architectures, specially designed hardware and software modules are available to improve throughput and time performance. Designers can improve the system performance and feasibility of algorithms utilizing these accelerators.

Although AI/ML algorithms implemented at the HDL level can provide higher performance, having high-level languages such as OpenCL, Python, and C/C++ for IP core design offers greater design flexibility. They also provide reduced design complexity and time by automating the AI/ML accelerator IP core synthesis process [272]. With the advent of the HLS software framework of Xilinx Vitis, it is now possible to use C/C ++-based high-level software languages for FPGA hardware design of AI/ML algorithms. The coexistence of PL and the PS in modern Xilinx SoCs offers users a wide range of optimization opportunities in the hardware design of AI/ML algorithms. Usual optimization approaches for the hardware design of AI/ML algorithms include systolic array architecture [273–275], roofline model [276], loop unrolling [277], and pipeline genera-

tion [278, 279].

In addition to these approaches, combining methods such as efficient use of on-chip memory, data quantization, and acceleration interface optimizations further help improve the performance of hardware accelerators [280]. In AI/ML applications requiring continuous memory access, choosing a CPU with a DMA feature is crucial to obtain high bandwidths that reduce the time it takes to access memory and improve performance. In addition, CPUs with built-in DSP blocks and dedicated hardware multiplier peripherals can be beneficial from the performance point of view as these can reduce the time required for calculations and data pre-processing.

Alternatively, designers can also prefer GPUs that offer parallel processing of data. GPUs have the advantage of performing more than one calculation simultaneously thanks to their scalable architecture in AI/ML applications as training of the algorithm in which a high workload with thousands of parameters needs to be optimized. However, the advantages provided by GPUs come with an inherent cost: power consumption. Because of their inborn high energy consumption, they are not a good fit for battery-powered mobile applications. On CPU-based platforms, designers have limited alternatives for optimization, like more memory allocation, more CPU cores, or changing the compiler options. However, parallel programming is only possible with multi-core-based processor architectures, which, in turn, increases the cost [281, 282].

Another optimization approach might be the hybrid use of a tight coupling of the CPU and GPU to have a balanced performance optimization [281, 283–285]. In such a scenario, while routine processes run on the CPU, the GPU might address the performance-demanding workloads. Dedicated AI/ML hardware solutions in ASIC form need to be more flexible and come with increased design costs and design time overheads. Hence, this feature renders them feasible only for applications with large quantities.

3.4.3 Software Optimization Techniques

Convergence principles are commonly used in ML/DL models, where data is fitted to the model. It is still debatable whether this method will lead to artificial general intelligence (AGI) [286]. On the other hand, DL currently delivers the best answer and achieves this goal using software optimization approaches. We can classify software optimization

methods as first-order, high-order, or derivative-free. [287].

We frequently encounter first-order optimization methods with gradient descent and variants in DL models. Gradient descent is a first-order optimization method. The gradient descent optimization method (batch gradient descent) is one of the oldest and most widely used optimization techniques [288]. This method involves iteratively updating variables in the (opposite) direction of the objective function's gradients. I can carry out the update so that the objective function gradually approaches the optimal value. The global minimum value is the best solution when the objective function is convex. Iterations must be performed carefully if the variables are close to their optimal values. The convergence process to the solution will be slower. This is because all training data is used at each iteration step.

Studies [289, 290] have suggested several parallelization methods to reduce the computational intensity. We also need to consider cost when dealing with large-scale data. Studies [291, 292] proposed SGD because batch gradient descent has high computational complexity at each iteration in large-scale data and does not allow online updating. SGD decreases the update time for managing enormous quantities of samples and eliminates a specific measure of computational redundancy, which seriously quickens the computation. Studies in [293–295] have shown that SGD achieves optimal convergence speed for strongly convex problems.

The adaptive learning rate method is another first-order optimization method [296]. The learning rate (LR) is one of the hyper-parameters for optimization. The LR determines whether the model should consider specific parts of the data. When LR is high, the model misses the details in data, whereas when LR is low, the model can access the finer details. The disadvantage of a low LR value is that the model runs slowly. Because LR is critical for the SGD, adaptive methods for accurately determining its value have been developed. Variants of SGD with adaptive LR such as Adaptive Moment Estimation (Adam) [297], Root Mean Squared Propagation (RMSProp) [298], Adaptive Gradients (AdaGrad) [299], Momentum [300], and Stochastic Approximation Method [301] are widely used in DL/ML (especially DNN) models. The gradient approach is preferred in some optimization problems because the derivative of the objective function may not exist or is difficult to estimate. This is where the derivative-fee optimization method comes

into play [302]. Instead of generating a systematic solution, a heuristic algorithm is used in this optimization method, which selects well-working methods such as particle swarm optimization and genetic algorithm.

Although there are numerous studies on optimization methods in the literature, in our work, we focused on the ones that have gained popularity recently and are frequently used in A/DAS applications. Designers should select optimization methods tailored to their problems based on data type, network size, and several model parameters. The SGD optimization method produces successful results in a binary or multi-class classification problem when using floating-point arithmetic for the model's weight and activation functions. However, if we use a fixed-point or quantized data type, the Adam optimization method brings out successful results [303–306].

We need to compute the error or the loss function to estimate the total model accuracy loss at the end of the training session. In addition, we need to constantly trace the instantaneous error value calculated by updating the weights on each evaluation step during training. We can select a loss function appropriate for the classification problem, such as a binary, multi-class, or multi-label loss function. We also use the loss function to determine the output layer configuration. If the problem only has two distinct classes, binary cross-entropy with a sigmoid activation function should be available at the output layer. Otherwise, if the problem is multi-class, categorical cross-entropy should be used with a Softmax activation function at the output layer. For binary classification, we can also use categorical cross-entropy. However, we must convert the data we classify here to the categorical (one-hot encoding).

If we try to use categorical cross-entropy to solve a problem that can be solved with binary cross-entropy, we will put additional strain on our hardware. This is due to one-hot encoding memory requirements, which is undesirable from an optimization point of view. Using cross-entropy as the error function in a classification problem with many classes (e.g., thousands) can be challenging. This is because each training sample contains a one-hot encoded vector with thousands of zero values, which require a large amount of memory. Sparse cross-entropy does not necessarily entail the target variable being encoded as one-hot before training. Sparse categorical cross-entropy is also encoded as normal integers. If each sample belongs directly to one class (mutually exclusive), we rec-

ommend the usage of sparse categorical cross-entropy. Categorical cross-entropy might be beneficial if a sample has more than one class or label so that we can save valuable resources such as time and memory. This approach is further made clear by the categorical cross-entropy loss function defined in (15) as;

$$Loss = -\sum_{i=1}^{N} y_i \log(\hat{y}_i) \tag{15}$$

Where y_i is the true label and $\hat{y}_i$ is the predicted label. Consider a thousand-class classification, with each sample belonging to only one class. Instead of adding up the number of classes for each sample, we use sparse categorical cross-entropy, which requires only one log operation. In addition, we use only one integer instead of a floating point in multi-class classification. In both cases, (15) is the same. As a result, they do not affect the accuracy of the inference. However, in terms of computational density, sparse-cross-entropy is preferable.

3.4.4 Compression

Compressing DNNs for resource-constrained devices mandates software and hardware optimization. As a result, researchers in the field have developed various methods to address the problem of performance and resource optimization. The compression of DNNs via size reduction and network pruning is presented in [307–311]. This method has three stages: the first stage involves learning essential connections while training a NN; the second stage entails applying the threshold to the weights to obtain the focused parts; and the third stage necessitates removing the neurons that are least significant.

The obtained network is then retrained. Compressing the network at the end of the retraining process will be more effective than doing it at the beginning. Data quantization and bit-width manipulation are two more key compression approaches [309–316]. The quantization method aims to reduce the number of bits in the network's weights and activation functions. Using low-precision arithmetic, one can compress the mesh by lowering the required weight bits. As a result, this will reduce the network's memory footprint and computational workload. The quantized binary NN topology stores weights with a single bit rather than a fp32, thus significantly reducing the required storage

space [263]. It is preferable to perform operations such as pruning in models. In this way, it could be possible to fit the NN-based AI/ML model designs onto FPGA devices with limited resources. Designers consider the time overhead due to the compression and decompression of data during the design process. It may introduce latency and prevent the use of this technique in performance-demanding AI/ML applications.

3.4.5 Aim-Oriented Design

Designers should adopt a goal-oriented approach that considers hardware, peripheral units, and software criteria when planning a study. Engineers must develop methods or algorithms adaptable to different platforms with minimal modifications. This flexibility should incorporate considerations of cost and performance of the equipment, aiming to achieve the desired accuracy of outputs using hardware that balances cost-effectiveness with suitable performance. Designers must carefully select input and output parameters in studies with resource constraints. Minimizing the number of inputs can help reduce the complexity of the outputs, which is essential when resources are limited. Despite a reduction in inputs, designers must still ensure that the system achieves the desired objectives. Maintaining algorithm simplicity is critical to enhancing system efficiency. Consider, for example, optimizing a classification problem where cross-entropy is used as a loss function. By assigning probabilities to each class and modeling the problem accordingly, this loss function aids in steering the probabilities closer to the correct class and away from incorrect ones. This approach addresses the classification challenge and transitions the focus towards optimizing the problem. By integrating the model with the optimization infrastructure already in place, the designer can effectively navigate to the project's next steps. This seamless integration ensures that hardware considerations and algorithmic complexities are aligned to meet the project's goals efficiently.

3.4.6 Tests for Optimal Solution

Testing a study is the primary tool to see the system's behavior. No one can make assumptions or corrections until a test execution. Engineers can optimize the system by creating software or hardware changes during the test period. However, in a study where everything is perfect in theory, designers might encounter unexpected problems during the

implementation phase. Hence, we can achieve optimization by testing a system to work optimally. A model behaves differently on different hardware platforms. Considering the constraints of the study, a designer should use hardware and software that provide minimum cost and maximum performance.

The energy consumption is directly proportional to the clock frequency. The higher the clock frequency, the higher the energy consumption. In an application, if the desired output occurs before the deadline and the power consumption is high, the energy consumed can be reduced by reducing the clock frequency. Designers can compare the application's energy consumption by running it on parallel and serial platforms. If both energy consumptions are identical, the designer can choose hardware capable of parallel processing. This equipment can comfortably perform the desired works at low frequencies without a heating problem. Heating is another factor that increases energy consumption. Designers can prevent heating problems with experimental tests. A metal heat sink or fan coolers might work. But here, the use of energy can increase with fan coolers. In this case, the efficiency should be checked by performing the case tests with and without a cooler. The model can be optimally operated, and energy can be saved by conducting similar tests.

4 Optimizing Performance: Hardware Acceleration Across Diverse Embedded Platforms

This section offers a comprehensive analysis of the embedded platforms that have been instrumental in my Ph.D. research and discusses the practical projects conducted using these platforms. It underscores the practicality of hardware acceleration, explicitly focusing on the reconfigurable architectures of SoC and MPSoC-FPGA, known for their ability to parallelize processes. The discussion starts with the PYNQ-Z1 development board, detailing the PYNQ: Python Productivity and FINN experimental frameworks employed in this board. Moreover, the section includes a comparative analysis, leveraging existing literature on hardware accelerators, software methods, Hw-Sw co-design, and numerous datasets to illuminate the strengths and weaknesses of the research performed.

4.1 PYNQ-Z1 Applications

4.1.1 Performance evaluation of real-time video processing edge detection on various platforms

This study details the design of a hardware accelerator for real-time edge detection in video streams. While various edge detection algorithms are available, this study favors explicitly using the Laplacian high-pass filter. The Laplacian of an image acts as a second-order or second-derivative enhancement technique, specifically designed to highlight delicate details within an image [317]. A Laplacian operator emphasizes sharp, abrupt transitions, making it a valuable tool in image processing. The Laplacian approximates the second derivative as a differential operator, as expressed in (16). It computes the Laplacian $F(x, y)$ for an image with pixel intensity values $I(x, y)$. It enhances image features by detecting sudden changes in pixel values, which typically signify edges or other critical details in an image.

$$F(x, y) = \nabla^2 f(x, y) = \frac{\partial^2 I(x, y)}{\partial x^2} + \frac{\partial^2 I(x, y)}{\partial y^2} \tag{16}$$

In technical terms, this study primarily focuses on grayscale images, which requires analyzing variations in the density of the image's pixels. A sudden increase or 'spike' in pixel density usually indicates the presence of an edge. Mathematically, this condition is

represented by taking the derivative of pixel intensity; an increase in intensity signifies an edge. The Laplacian filter detects edges in both the x and y dimensions, implying that we are computing the second derivative of the pixel densities. As shown in Figure 4.1, the graph intersects zero upon taking this second derivative. Thus, this methodology involves searching for zero-pixel points in the image and identifying these as edge locations. This technique allows us to highlight major features in an image at the edges.

It is necessary to interpret the equation in discrete time to apply the Laplacian operator in digital signal processing, as real-world digital signals are represented as a series of discrete values over time. The Laplacian operator, initially designed for continuous functions, is remarkably adaptable. Given that it involves a two-variable partial differentiation operation, we initially express it as a single-variable function and subsequently extend this to a two-variable formulation. Similar to (16), the Laplacian operator for a single-variable continuous function can be described as illustrated in (17). We are adapting the Laplacian equation to accommodate the discrete nature of digital signals, thereby facilitating its application in digital signal processing using (18).

$$\nabla f(x) = \frac{\partial^2 f(x)}{\partial x^2} \tag{17}$$

$$Lf(x) = \frac{f(x-h) - 2f(x) + f(x+h)}{h^2} \tag{18}$$

L represents the discrete Laplacian operator, and h denotes a constant. This discretization implies that the function $f(x)$ is sampled evenly at h intervals. As illustrated in (19), this methodology can be implemented convolutionally through software or hardware.

$$Lf(x) = l * f \tag{19}$$

In this case, l is a filter, and the Laplacian in (19) may be expressed as a matrix operator depending on the assumed boundary condition. Laplacian filter kernels exhibit a distinct pattern: they feature negative values along a cross-shaped center within the array, resembling a plus sign, while the corners typically hold zero or positive values. The central value of the array may be harmful or positive. For instance, a five-by-five kernel used in Laplacian filters has a specific configuration, as depicted in Figure 4.2. A kernel is a small

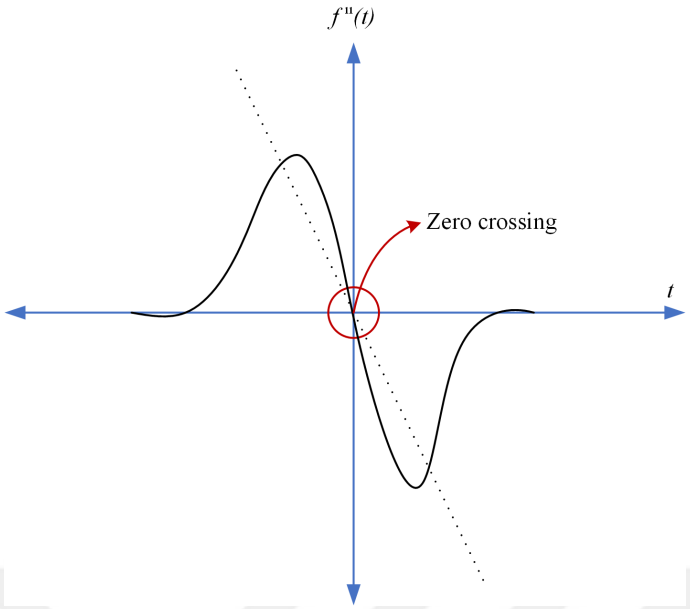


Figure 4.1: Reference graph of the pixels which goes to zero

matrix employed in image processing techniques such as blurring, sharpening, and edge detection, among others. The configuration of negative and positive values within the Laplacian kernel enables the accentuation of rapid intensity changes in images indicative of edges or intricate features.

0	0	-1	0	0
0	-1	-2	-1	0
-1	-2	16	-2	-1
0	-1	-2	-1	0
0	0	-1	0	0

Figure 4.2: A 5 by 5 kernel for a Laplacian filter

High-performance video processing is implemented in this study in an FPGA-based embedded SoC. Embedded vision technology, which is still in its developmental stages, relies heavily on FPGA and GPU processors integrated into embedded systems [318]. FPGA architectures have gained prominence over GPUs and CPUs for embedded vision applications in recent years [319].ASICs are known for their high performance in terms of power and speed. However, their production costs are high, and they need more flexibility, which limits their usage. On the other hand, FPGAs are faster, have lower latency, consume

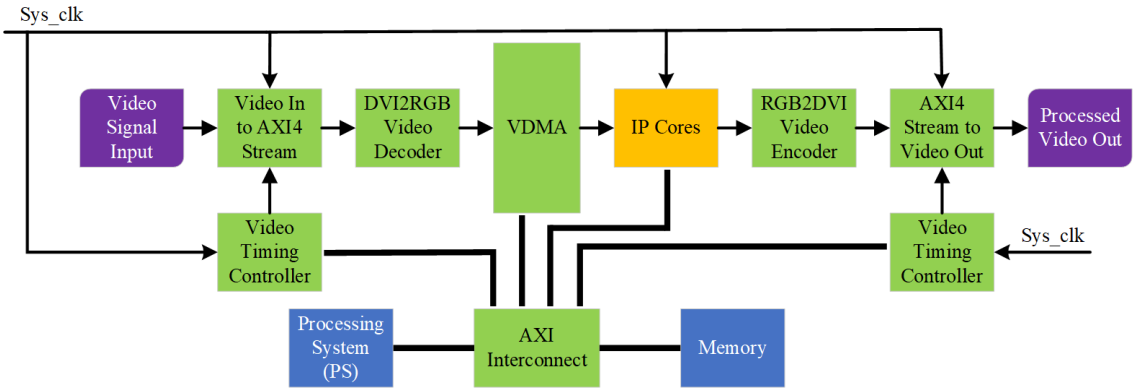


Figure 4.3: Simplified flow chart of video streaming steps

less energy than GPUs, and come with inherent parallel processing capabilities. Hence, they are an ideal choice for embedded vision. This has led to developing "FPGA-vision," distinct from the broader computer vision landscape.

The growth of FPGA-vision technology has been remarkable, especially with the introduction of ZYNQ architectures. ZYNQ integrates ARM-based processor programmability with FPGA hardware programmability, as shown in Fig 4.3 ZYNQ architectures are also compatible with high-level languages like C/C++ and Python. While HDL are familiar with FPGAs, they can slow the design process. High-level synthesis approaches are increasingly needed to overcome this bottleneck. High-level languages have been introduced into hardware design and synthesis to address this. This study applied this approach by creating a Laplacian high-pass filter design on the ZYNQ-7020 SoC for real-time video edge detection using the PYNQ-Z1. I tested the approach on different platforms: the SoC (hardware-assisted software), dual-core ARM Cortex-A9 (software only), and a personal computer (also software only) to evaluate effectiveness and system response speed.

4.1.1.1 Hardware and Software Design

The ZYNQ architecture is a SoC that combines a dual-core ARM Cortex-A9 PS with a PL integrated circuit (IC). This combination offers high-bandwidth and low-latency connections, utilizing the advanced extensible interface (AXI) standard [320]. During the design process, tasks can be allocated between the processor and FPGA sections according to the system's requirements. This stage's essence is highlighted by its impact on the program's speed and functionality being executed and the overall system performance. These outcomes are directly influenced by the division of tasks between the PS and PL.

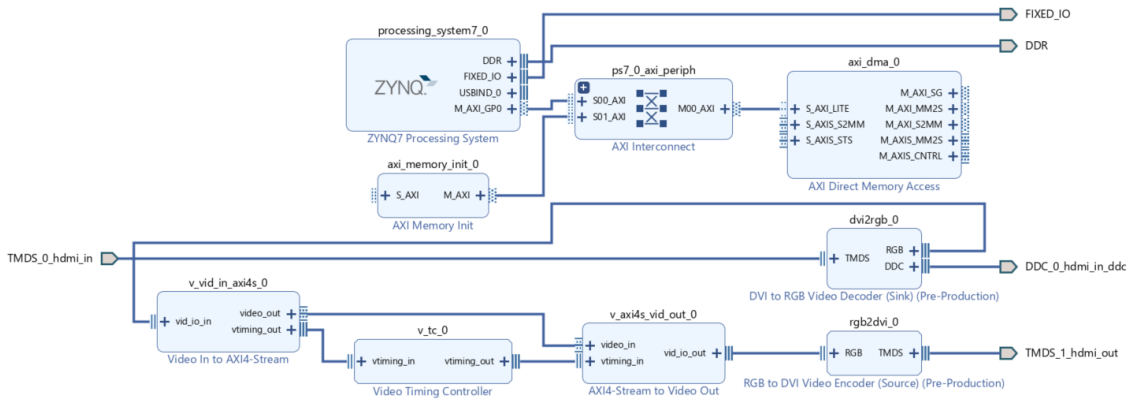


Figure 4.4: Basic interface view with the pipeline of computer vision overlay

In this research, I allocated the PL components that require high speed and computational density for video processing while the PS handled the other parts. I have defined the functional blocks in the hardware design and constructed them as Intellectual Properties (IPs), creating a connection between the PL and PS using suitable AXI-DMA interfaces. I used the Vivado 2020.1 integrated development environment for digital hardware development and high-level synthesis integrated system design in the PL. The design of a hardware accelerator in the PL environment involves creating an overlay composed of IP blocks. The PYNQ-Z1 development board is ideal for FPGA-based vision applications as it connects to PL pins for both HDMI-in and HDMI-out interfaces. The pipeline architecture shown in Figure 4.4 comprises these IP blocks and is a hardware accelerator for video processing applications.

Digilent provides some IP cores that are compatible with Xilinx Vivado IP and are free of charge [321]. However, since these IP cores are not built into Vivado, designers must manually integrate them. Once the pipeline architecture is constructed, it can be incorporated into the machine-vision overlay, accelerating and parallelizing video stream processing. The top layer of the pipeline design includes a 100MHz system clock, video decoder/encoder, video timing controller, high-performance AXI stream, and other supplementary IP blocks.

The FPGA is an excellent choice for matrix operations in convolution because of its pipeline's ability to complete tasks quickly. Convolution with a 5x5 kernel is computationally intensive, and the process takes longer since the video is in Full HD. The convolution operations are critical for video processing and are executed rapidly in the

PL before being transferred to the output via AXI-DMA. The elapsed time during the convolution process and data transfer represents the processing time of the PL.

The PYNQ-Z1 is a crucial component in this study, and it was programmed using Python. The PYNQ-Z1 connects directly to a Jupyter Notebook via a static IP address. Python coding is not used for the IP blocks created for hardware description and verification; instead, it plays a crucial role in facilitating the interaction between the PS and PL via interfaces.

For the video processing accelerator in the PL, a frame buffer stream architecture is defined using Python. The video pixel information is initially buffered to ensure a smooth data flow and then processed at the necessary bandwidth. Once processed, the processor transfers the data to the video processing pipeline architecture using AXI interconnects. The connection between the PS and the AXI stream is established with DMA. The maximum DMA transaction size at the outset is 14 bits or 16KB. For more extensive DMA operations, the bit width must be increased (up to 26 bits for ZYNQ-7000) during the design of Vivado IP.

I used three different software algorithms to compare their performances: standalone ARM (PS), a combination of FPGA and ARM (PL+PS), and a multi-core CPU. Figure 4.3 displays a flow chart of the operations performed on both the PS and PL sides. The specifications for the multi-core CPU can be found in Table 4.1, but please note that only one core is used despite the computer having four cores. Software parallelization techniques like Pthreads, MPI, or OpenMP are not implemented.

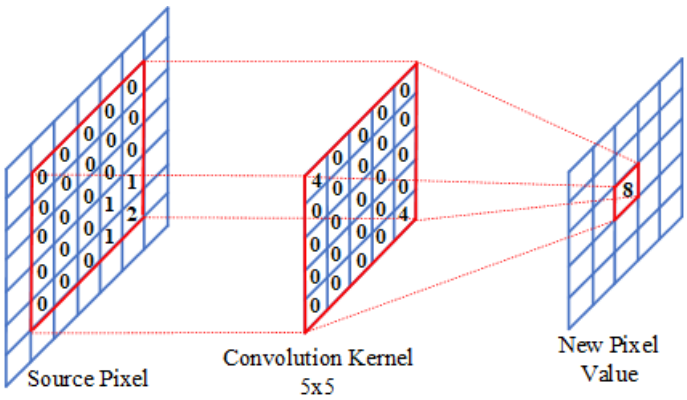


Figure 4.5: Convolution operation with 5x5 kernel

Table 4.1: Comparison of the 5x5 kernel Laplacian high-pass Filter's real-time video processing edge detection performance on various Hardware platforms

Platform	Embedding Methods	Hardware Architecture	FPS Value	Acceleration Rate	Cost ($)	Price / Performance ($ /×)
CORA Z7	Software	ARM Cortex-A9 Dual-Core 667 MHz	1.912	1×	147	147
PYNQ-Z1	Software + Hardware	ZYNQ - Z7020	44.214	**23.124**×	299	**12.921**
Personal Computer	Software	i7 7700 HQ 16 GB RAM Quad-Core 2.86 GHz	22.456	11.744×	~1000	85.149

Table 4.2: Post implementation of ZYNQ architecture resource usage in table

Resources	PYNQ Z1 Available	Utilization (%)	ZYNQ ZC702 Available	Utilization [322] (%)	ZYNQ Z-7010 Available	Utilization [323] (%)
LUT	53200	2.29	53200	10.6	17600	41.48
LUTRAM	17400	0.14	17400	N/A	N/A	N/A
FF	106400	1.64	106400	7.44	35200	34.97
BRAM	140	1.43	140	5	60	20
IO	125	16	200	26	100	32
BUFG	32	15.63	N/A	N/A	N/A	N/A
MMCM	4	50	N/A	N/A	N/A	N/A
PLL	4	25	N/A	N/A	N/A	N/A

4.1.1.2 Test and Evaluation

I first tested the ARM processor as a software accelerator to experiment. This setup achieved an FPS 1.912 following the convolution process using a 5x5 kernel. I then employed the ARM and FPGA architectures together under identical conditions, resulting in a significant increase in the FPS to 44,214. I displayed the real-time video processing application output on the SoC (FPGA+ARM) in Figure 4.7. I inverted the image to make it easier for readers to understand and analyze.

Finally, I measured the benchmark on a personal computer with a quad-core 2.8 GHz Intel i7 processor and 16 GB of RAM. Here, I achieved 22,456 FPS. Table 4.1 reveals that the ZYNQ 7020 SoC environment outperforms other environments regarding price and performance.

4.1.1.3 Results and Discussion

The study conducted a performance evaluation of real-time video processing for edge detection across three distinct hardware platforms: an ARM processor, an ARM+FPGA-based SoC, and a single-core Intel i7 processor. The ARM+FPGA-based SoC significantly outperformed the other platforms, executing the filtering algorithms 23.124 times faster than the ARM processor and 1.969 times faster than the Intel i7 processor, highlighting its superior parallel processing capabilities. This acceleration is crucial for applications where speed is crucial. The cost-performance

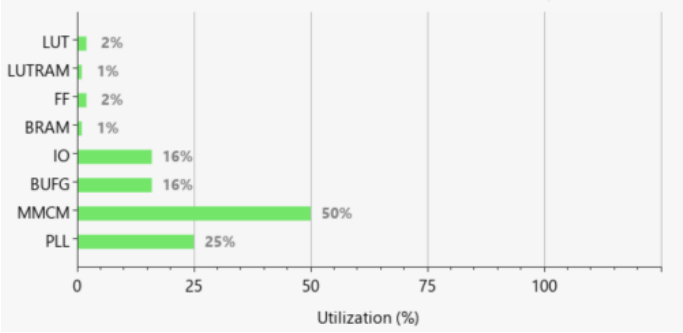


Figure 4.6: Post implementation of ZYNQ architecture resource usage in graph

analysis further emphasized the ARM+FPGA-based SoC as the most cost-effective option, with a compelling price-performance ratio of approximately $12.921 per acceleration unit, suggesting a strong case for its adoption in speed-critical environments. Resource utilization on the SoC was efficiently managed, as shown in Figure 4.6 and Table 4.2 of the document, where the hybrid system maintained reasonable use of resources such as LUTs and BRAMs, ensuring the system could handle higher workloads without excessive power consumption or heat generation. The combination of ARM processors with FPGA accelerators balances speed, cost, and efficiency, positioning it as an ideal choice for enhancing throughput in real-time video processing applications.



(a) Filtered image (b) Inverse of filtered image

Figure 4.7: ARM+FPGA-based accelerator for a Laplacian high-pass filter a) filtered image, b) inverse image

4.1.2 Hardware Acceleration of FIR filter Implementation on ZYNQ SoC

In signal processing applications, filters are critical in attenuating noise or undesired signals to isolate the signal of interest. These filters are categorized into discrete-time and continuous-time systems. Within discrete-time filters, there exist two main types:

finite impulse response (FIR) and infinite impulse response (IIR) filters. FIR filters are particularly favored in digital signal processing applications owing to their linear phase responses and inherent stability, despite the challenges of low computational speed and the requirement for high numbers of filter coefficients [324–326]. Hardware accelerators with inherently parallel processing capabilities, such as FPGAs, are increasingly utilized to mitigate the computational demands associated with FIR filters. The adoption of FPGAs in digital signal processing has grown over time, spurred by the development of platforms that support high-level programming languages, which offer an alternative to the more traditional low-level HDLs [327,328]. Using HDLs, such as Verilog and VHDL, in digital filter design can introduce significant delays in the design timeline, thereby creating bottlenecks. The transition to high-level synthesis approaches using languages like C, C++, and Python in hardware design and synthesis processes has been crucial in overcoming these delays.

In alignment with previous research, I selected the PYNQ-Z1 SoC/FPGA development board for this study. I developed a low-pass FIR digital filter specifically designed for real-time attenuation of high-frequency noise signals. To evaluate the effectiveness and system sensitivity of the implemented filter, I conducted comparative tests on different platforms: the SoC (with hardware-assisted software implementation), a dual-core ARM Cortex-A9 (solely software implementation), and a personal computer (PC, solely software implementation).

4.1.2.1 Discrete time FIR filter design A micro-controller and various peripheral units with memory are used to hold the filter coefficients. FIR filters generally have a finite time impulse response and consequently settle to zero in the finite time. These filters are systems without feedback. Therefore, transfer functions do not have poles; they only contain zeros. Generally, filters are signal conditioners that allow AC components and block DC components of a function. The mathematical equation (20), (21), (22) and the block diagram Figure 4.8 of the nth order FIR filter has given.

$$y[n] = \sum_{k=0}^{M} b_k x[n-k] \tag{20}$$

$$H(z) = \sum_{k=0}^{M} b_k z^{-k} \tag{21}$$

$$H(z) = b_0 \prod_{k=1}^{M} (1 - c_k z^{-1}) \tag{22}$$

where b_k is the reverse filter coefficients; $x[n]$ is the input signal; $y[n]$ is the output signal; M is the filter order. The design of FIR filters diverges from that of IIR filters

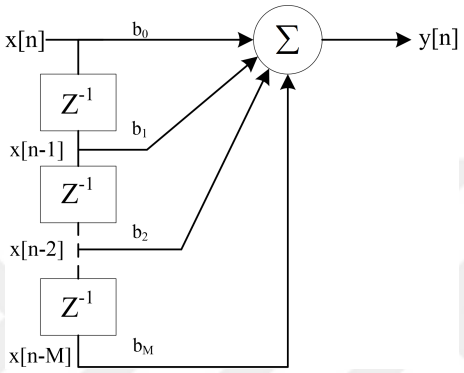


Figure 4.8: FIR Filter Block Diagram

due to differing design methodologies, particularly the use of window functions such as Rectangular, Hamming, Blackman, Bartlett, and Hann. These windows shape the filter's frequency response, influencing parameters like the peak approximation error, which varies with the window type and determines the stop-band attenuation. In the FIR filter design shown in Figure 4.9 using the Hamming window, parameters δ_1 and δ_2 represent deviations from the ideal response. At the same time, ω_p and ω_s denote the pass-band and stop-band corner frequencies, respectively, with ω_c as the critical frequency of the filter. This approach ensures that the filter meets the required specifications with effective frequency attenuation. The filter with sampling frequency $f_{sa} = 10MHz$, $f_s = 2.5MHz$, $f_p = 2MHz$, minimum stop band attenuation -40dB.

$$w_p = 2\pi \frac{f_p}{f_{sa}} = 2\pi \frac{2MHz}{10MHz} = 1.256 \frac{rad}{sec} \tag{23}$$

$$w_s = 2\pi \frac{f_s}{f_{sa}} = 2\pi \frac{2.5MHz}{10MHz} = 1.57 \frac{rad}{sec} \tag{24}$$

Transition band $w_s - w_p = 0.5\pi - 0.4\pi = 0.1\pi$. Main lobe width of the Hamming window is given $\frac{8\pi}{M}$ and M=40 (see Tab.4.3), so the designed filter length is 41. Critical frequency

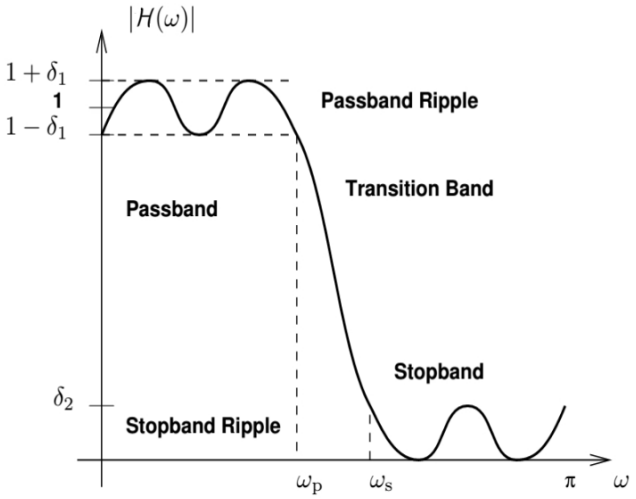


Figure 4.9: Low pass filter frequency response [5]

of the filter to be designed is obtained by (25):

$$w_c = \frac{w_s + w_p}{2} = \frac{0.9\pi}{2} = 0.45\pi \tag{25}$$

Low pass filter motivation reaction is given by backwards discrete-time Fourier transform.

Table 4.3: Window functions and their properties [18]

Window Type	Peak Sidelobe Amplitude (Relative dB)	Approximate Width of Main lobe	Peak Approximation Error 20log(δ) (dB)
Rectangular	-13	$\frac{4\pi}{M-1}$	-21
Bartlett	-25	$\frac{8\pi}{M}$	-25
Hann	-31	$\frac{8\pi}{M}$	-44
Hamming	-41	$\frac{8\pi}{M}$	-53
Blackman	-57	$\frac{12\pi}{M}$	-74

$$h_d[n] = \frac{1}{2\pi}\int_{+w_c}^{-w_c} H_d(w)e^{jwn}dw = \frac{Sin(w_c n)}{n\pi} \tag{26}$$

where, $h_d[n]$ is ideal impulse response.

The Humming window equation (ω_n) is defined as follow:

$$\omega[n] = \begin{cases} 0.54 + 0.46cos(\frac{2n\pi}{M}) & 0 \leq n \leq M \\ 0 & \text{otherwise} \end{cases} \tag{27}$$

$$h_{d,lowpass}[n] = \frac{w_c}{\pi}Sinc(\frac{w_c n}{\pi}) = 0.45Sinc(\frac{45n}{100}) \tag{28}$$

Multiply hamming window (w_n) by $h_{d,lowpass}[n]$ and apply $M/2$ time shift (29).

$$h[n] = [0.54 + 0.46cos(\frac{2n\pi}{M})][0.45sinc(\frac{45n-20}{100})]...0 \leq n \leq M \tag{29}$$

Where; $h[n]$ is the impulse response of the FIR filter.

4.1.2.2 Hardware and software Design In the design phase, the distribution of tasks between the PS and PL sections is crucial for optimizing the speed and functionality of the program, as the overall performance of the system hinges on this task allocation. Functional blocks within the hardware design were organized as IPs to establish the PL-PS connection using suitable AXI and DMA interfaces. The design environment used was the Vivado 2021.1 integrated development environment, which supports digital hardware development and high-level synthesis in the PL. The hardware accelerator, termed the 'overlay,' comprises various IP blocks, including the FIR compiler IP block configurable to meet specific application needs. The coefficients for the FIR filter were sourced from an online open-source application [329], indicating the overlay's adaptability (Figure 4.12 illustrates the overlay design, and Figure 4.13 shows the low pass FIR filter).

During the processing, data sampled from the input signal and the computed filter coefficients are rapidly handled in the PL environment and then transferred to the output ports using AXI direct memory access, showcasing the PL's processing efficiency. The high-level Python programming language was utilized on the PYNQ-Z1 SoC development board for software operations, interfaced through Jupyter Notebook. The input test signal, a composite of three different sinusoidal signals with varying amplitudes and frequencies, is sampled at 10MHz on the PS side to generate the digital data for the PL. After mathematical processing, the system ensures that the resultant coefficients fall within the desired frequency range before relaying to the output through the AXI interface. A flowchart detailing the software operations and the sequence of filter coefficients is presented in Figure 4.10. The design achieves rapid computation capabilities in the FPGA section, significantly outpacing the ARM processor, which is crucial for handling high-speed signal processing tasks.

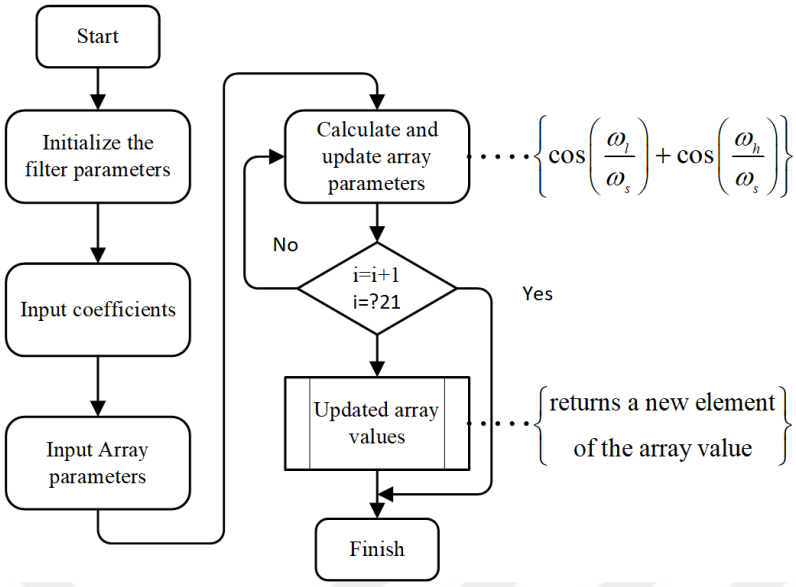


Figure 4.10: Flowchart of obtaining the signal of interest.

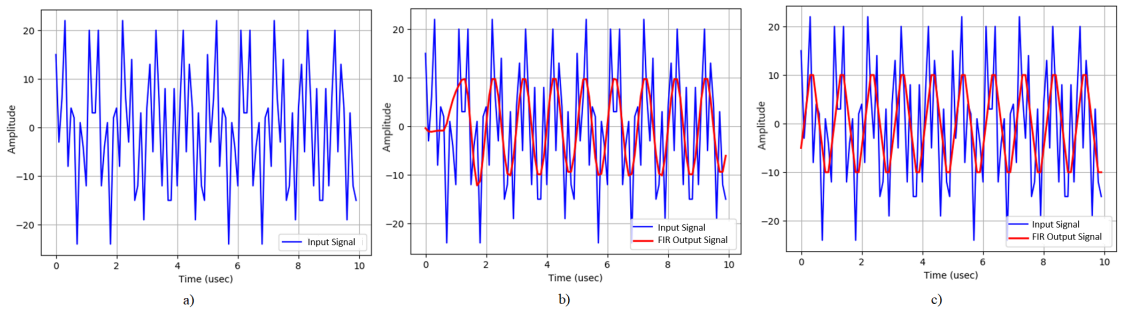


Figure 4.11: Operation of the low pass FIR filter; a) Noisy input signal, b) The filtered signal on the bare ARM processor, c) The filtered signal on ARM+FPGA hybrid use

Table 4.4: Performance comparison of FIR filter implemented in different design environments

Platform	Embedding Type	Hardware Architecture	Execution Time (ms)	Acceleration Rate (×)	Cost ($)	Price / Performance ($)
PYNQ-Z1	Software	ARM Cortex A-9 Dual-Core 650 MHz	45.1	1 ×	260	260
PYNQ-Z1	Hardware + Software	Zynq -7020 SoC	5.09	8.86 ×	260	**29.34**
Personal Computer	Software	i7 7700 HQ 16 GB RAM Quad-Core 2.86 GHz	2.56	17.61 ×	1000	56.78

4.1.2.3 Test and Evaluation Results In this research, I evaluated the performance of low-pass FIR filters in three environments similar to the previous work: on an ARM processor only, using a hybrid ARM and FPGA setup, and on a standard PC. Figure 4.11-a shows the input signal, which is a composite of three sinusoidal signals with different amplitudes and frequencies: 10V, a 1MHz signal of interest, and two noisy signals of 15V at 3.6MHz and 20V at 5MHz. The output of the FIR filter, which effectively reduces the

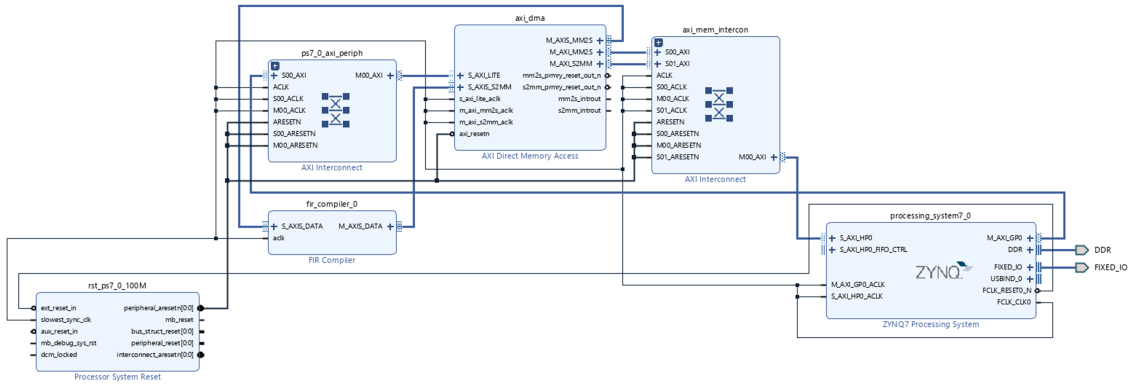


Figure 4.12: Hardware accelerator overlay design.

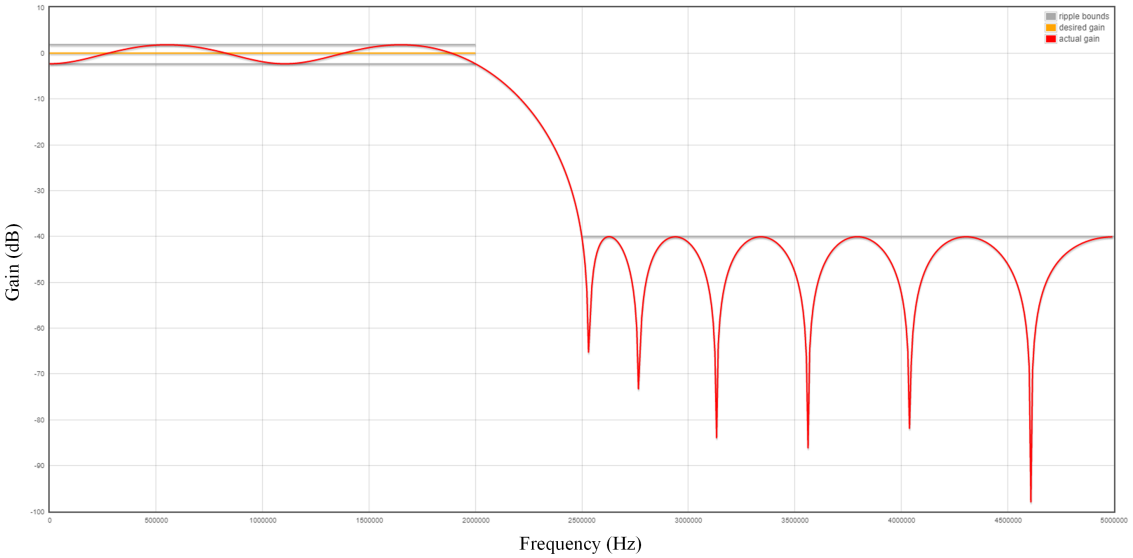


Figure 4.13: Low pass FIR filter with 2MHz cutoff frequency.

noisy components and preserves the signal of interest, is shown in Figure 4.11-b. This filter was first implemented on an ARM processor and took approximately 45.1 ms to obtain the desired signal.

Further tests include the approach using the FPGA and ARM processor. Figure 4.11-c shows this setup's input and output signals. The data communication between the two units was managed by AXI-DMA, resulting in a shorter processing time of 5.09 ms. Although the AXI interface provides high bandwidth and low latency, there is some inherent delay in data transmission and reception. However, this delay is usually negligible in applications requiring very high data density, making hybrid architectures particularly advantageous.

Finally, I ran the filter on a PC with a quad-core 2.8GHz Intel i7 processor and 16GB of RAM. The processing time in this environment was significantly faster at 2.56ms. The

comparative performance and computation times for these environments are summarized in Table 4.4. From the analysis, it is clear that the hybrid use of the Zynq 7020 SoC offers superior price-performance efficiency compared to other test environments and underlines the benefits of integrating software and hardware in complex signal processing tasks.

4.1.2.4 Conclusions & Discussion FIR filters are characterized by their linear phase properties and finite impulse response, devoid of feedback mechanisms. These attributes make them highly favored in numerous signal processing applications, particularly where linear phase characteristics are crucial, and processing needs to be efficient. This study capitalized on these advantages when designing a low-pass FIR filter for scenarios demanding high processing speed. I tested the filter by applying a composite noisy signal $(10\sin(10^6 \times 2\pi t) + 15\sin(3.6 \times 10^6 \times 2\pi t) + 20\sin(5 \times 10^6 \times 2\pi t))$ at the input and successfully isolated the signal of interest $(10\sin(10^6 \times 2\pi t))$ at the output. The filter's performance was evaluated across different hardware platforms, and the outcomes are documented in Table 4.4.

According to Table 4.4, while a personal computer offers the fastest processing, it is less cost-effective and suitable for portable or battery-powered devices. The study reveals that a hybrid approach, integrating hardware and software, offers a more balanced solution in terms of performance, cost, and flexibility. This hybrid design strategy diverges from traditional approaches by enabling optimization of both hardware and software components. I used the Vivado Design Suite 2021.1 for the hardware design to create a pipeline architecture. This architecture was designed to facilitate rapid and parallel mathematical operations while the PS efficiently handled non-computational tasks. The result was a heterogeneous computing environment. Such hybrid architectural studies integrating hardware and software are increasingly considered essential in addressing contemporary challenges. Only reliance on either software or hardware is often inadequate due to the limitations of semiconductor process technology, which only allows for incremental improvements.

4.1.3 Real-time Hardware Accelerator of Low Precision Quantized Custom NN Model on ZYNQ SoC

In this study, I employed the FINN experimental framework [260, 330], which provides an integrated process for exploring and implementing QNN inference solutions on FPGAs. FINN specializes in generating dataflow architectures that physically represent the network implemented in space. It focuses on a co-design approach and design space exploration specifically for tuning quantization and parallelization to optimize solutions based on specific resource and performance requirements rather than serving as a generic DNN acceleration solution.

Throughout the development of this project, I explored the relationship between speed and inference accuracy, aiming to enhance the method previously described in [331]. The objective was to improve inference accuracy, and to this end, I conducted a performance comparison of the network after configuring the model's weights and activations to low-precision INT2. The model was constructed using the PyTorch framework, and I utilized the Brevitas [332] library for quantization. Brevitas is explicitly designed for PyTorch to facilitate QAT, which aims to minimize the impact of data loss during the NN training, thereby preserving the model's predictive accuracy during inference.

Figure 4.14 illustrates the QAT training sequence flow, highlighting the process of quantizing weight and activations to INT2 both before and after the process. The model maintains the fFP32 format after the convolution layer and the bias application. To achieve an INT2 output for the model, I utilized the QuanReLU function. This method emphasizes maintaining a balance between computational efficiency and model performance. By applying the QuantReLU method at the output stage, I could execute quantization according to the desired integer-bit value. The quantized function is characterized by specific equations detailed in [333].

To deploy the quantized model on FPGA hardware platforms, tools such as Onnxruntime [334], FINN [260, 335], or PyTorch's quantized inference operators were used. This method ensures that NN solutions are optimized for specialized hardware, creating an efficient and customized implementation.

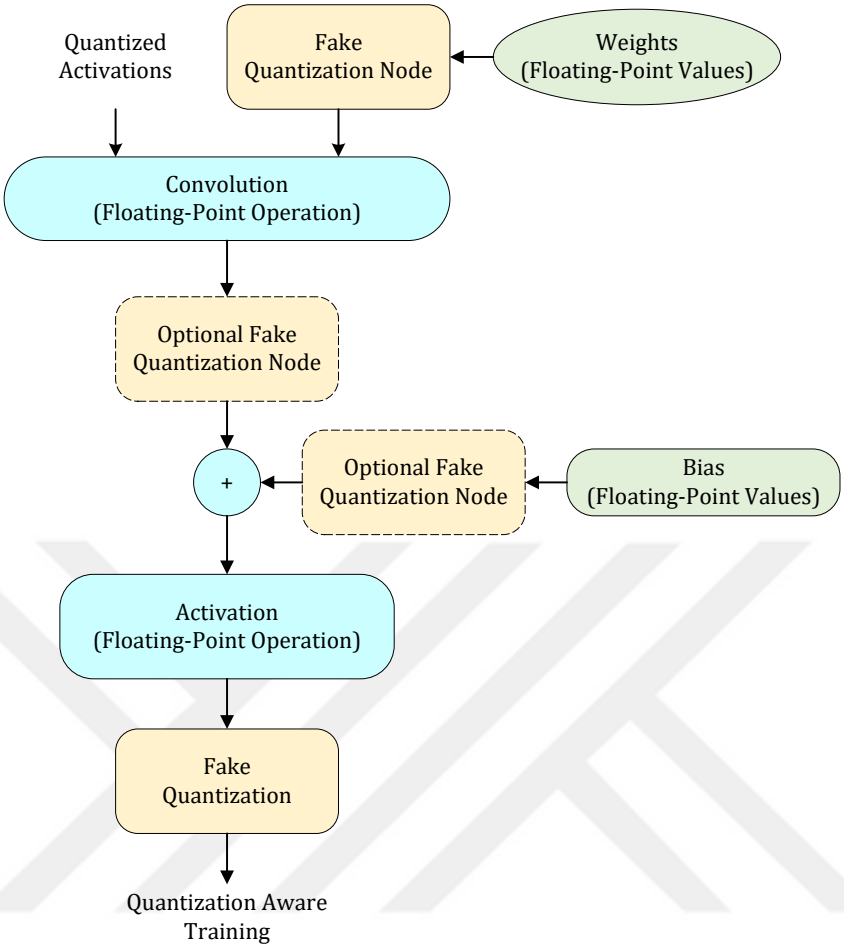


Figure 4.14: A simplified representative QAT

$$QuantReLU_{(x,z_x,y_x,k)} = \begin{cases} z_y & if\ x < z_x \\ z_y + k(x - z_x) & if\ x \geq z_x \end{cases} \tag{30}$$

When $z_x = 0$, $z_y = 0$, and $k = 1$, the well-known ReLU function, commonly employed in DL models, surfaces as a particular case of the given definition. This highlights the versatility and general applicability of our quantization approach

$$ReLU_{(x,0,0,1)} = \begin{cases} 0 & if\ x < z_x \\ 1 & if\ x \geq z_x \end{cases} \tag{31}$$

Subsequently, we shall examine the mathematical computation of QuantReLU.

$$
\begin{aligned}
y &= ReLU(x, 0, 0, 1) \\
&= \begin{cases} 0 & if\ x < z_x \\ 1 & if\ x \geq z_x \end{cases} \\
&= s_y(y_q - z_y) \\
&= ReLU(s_x(x_q - z_x), 0, 0, 1) \\
&= \begin{cases} 0 & if\ s_x(x_q - z_x) < 0 \\ (s_x(x_q - z_x) & if\ s_x(x_q - z_x) \geq 0 \end{cases} \\
&= \begin{cases} 0 & if\ x_q < z_x \\ s_x(x_q - z_x) & if\ x_q \geq z_x \end{cases}
\end{aligned} \tag{32}
$$

Accordingly;

$$
\begin{aligned}
y_q &= \begin{cases} z_y & if\ (x_q < z_x \\ z_y + \frac{s_x}{s_y}(x_q - z_x) & if\ x_q \geq z_x \end{cases} \\
&= ReLU(x_q, z_x, z_y, \frac{s_x}{s_y})
\end{aligned} \tag{33}
$$

Therefore, to carry out the QuantReLU operation that corresponds to the floating-point $y_q = ReLU(x, 0, 0, 1)$, it simply necessitates the execution of the following:

$$
y_q = ReLU(x_q, z_x, z_y, \frac{s_x}{s_y}) \tag{34}
$$

In this context, x_q represents quantized matrices, while z_x and z_y denote zero points, and s is a positive floating-point scale.

4.1.3.1 Setup, Training and Implementation of the Model

I extended the foundational work presented in [331] by incorporating three additional hidden layers into the NN model. After each layer, I quantified the weight and activation functions to INT-2 precision. Subsequently, I integrated `BatchNorm1d` and `Dropout` layers to enhance the

model's generalization and mitigate over-fitting. The training was conducted over 100 epochs with a batch size of 100, which markedly enhanced the model's performance during training and subsequent inference phases.

For the classification task, I employed the `BCEWithLogitsLoss` cost function, designed to output a probability value between 0 and 1 for each class. This loss function synergistically combines binary cross entropy (BCE) loss and a logistic sigmoid activation function to enhance numerical stability during training. It accepts two tensors, one representing the logits and the other the corresponding labels, and computes the BCE loss between these two tensors.

Unlike using a standalone BCE loss after a sigmoid function, `BCEWithLogitsLoss` [336] ensures improved numerical stability, more efficient gradient computation, and better memory utilization. The loss function automatically applies the sigmoid activation to the logit tensor x and calculates the BCE loss with the target tensor y. The model optimization was carried out using the Adam optimizer, renowned for its effectiveness in handling sparse gradients and adaptive learning rate adjustments, facilitating faster and more reliable convergence of the training process.

The logistic sigmoid function, denoted by $\sigma(z)$, is a commonly used activation function in NNs, particularly for binary classification tasks. It maps any real-valued number into the interval $[0, 1]$, making it appropriate for converting logits into probabilities. The function is defined as follows:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{35}$$

where: z is the input to the function, which can be any real-valued number or a tensor of real-valued numbers.

For tensors x and y, where x represents the logits and y the labels, both having identical shapes, and n represents the number of elements in each tensor, the logistic sigmoid function can be applied to each element x_i of tensor x to produce a new tensor of probabilities:

$$\sigma(x_i) = \frac{1}{1 + e^{-x_i}}, \quad \text{for each } i \text{ in } 1, 2, \ldots, n \tag{36}$$

This transformed tensor $\sigma(x)$ can then be used to compute the binary cross-entropy loss with respect to the labels y, for all elements in the tensors. The binary cross-entropy loss

for a single prediction and true label is given by:

$$\text{BCE}(x_i, y_i) = -\left[y_i \cdot \log(\sigma(x_i)) + (1 - y_i) \cdot \log(1 - \sigma(x_i))\right] \tag{37}$$

To compute the total loss over the entire dataset, you average the binary cross-entropy loss over all n elements:

$$\text{Loss}(x, y) = \frac{1}{n} \sum_{i=1}^{n} \text{BCE}(x_i, y_i) \tag{38}$$

This loss function measures the performance of the classification model, where a lower loss indicates a better model that predicts probabilities closer to the actual binary labels in y. The `Adam` optimization algorithm is designed to calculate distinct adaptive learning

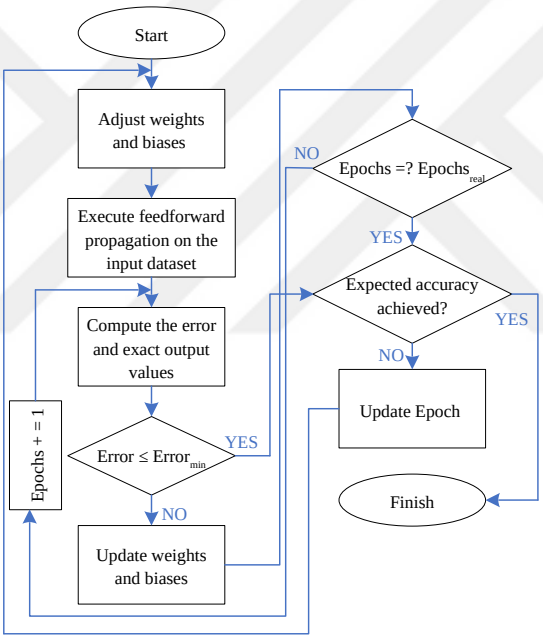


Figure 4.15: Algorithmic flowchart of the model

rates for various parameters from the approximations of the first and second moments of the gradient vectors. It operates by maintaining the exponentially decaying averages of past gradients, denoted as $\hat{m}_t$, and the squared gradients, denoted as $\hat{v}_t$, for each parameter. These values are updated at each iteration according to the following equations:

$$\hat{m}_t = \beta_1 \hat{m}_{t-1} + (1 - \beta_1) g_t, \tag{39}$$

$$\hat{v}_t = \beta_2 \hat{v}_{t-1} + (1 - \beta_2) g_t^2, \tag{40}$$

where:

- g_t represents the gradient at time step t,
- β_1 and β_2 are the decay rates that control the moving averages of the gradient and its square, typically set to values close to 1.

The updates to the parameters at each iteration are then governed by the adjusted learning rates as follows:

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{v}_t} + \epsilon}\hat{m}_t, \tag{41}$$

where:

- θ_t is the parameter vector at iteration t,
- η is the learning rate,
- ϵ is a small scalar added to improve numerical stability.

This method ensures that each parameter receives an individual adjustment of its learning rate, allowing for a more efficient convergence, particularly in scenarios involving sparse gradients or complex objective functions.
I followed the algorithmic sequence shown in Figure 4.15 to train the NN model displayed in Figure 4.16. The first step in the flowchart involved randomly setting the model's weights and activation functions. After this, I conducted feed-forward propagation using the input dataset. This step was followed by an evaluation to assess the difference between predicted and actual outcomes. The training process was stopped when the error reached the lowest possible value. However, if the error exceeded the predefined acceptable range, the weights were updated to correct the training direction. I increased the epoch count to ensure thorough model training if more than the number of epochs completed was needed to achieve satisfactory results.

4.1.3.2 Performance Analysis CNNs and DNNs are widely utilized in artificial intelligence to address various challenges. The decision to use either a CNN or DNN and the corresponding level of simplicity or complexity of the model depends on the specific requirements and complexity of the problem being addressed. Basic CNN architectures

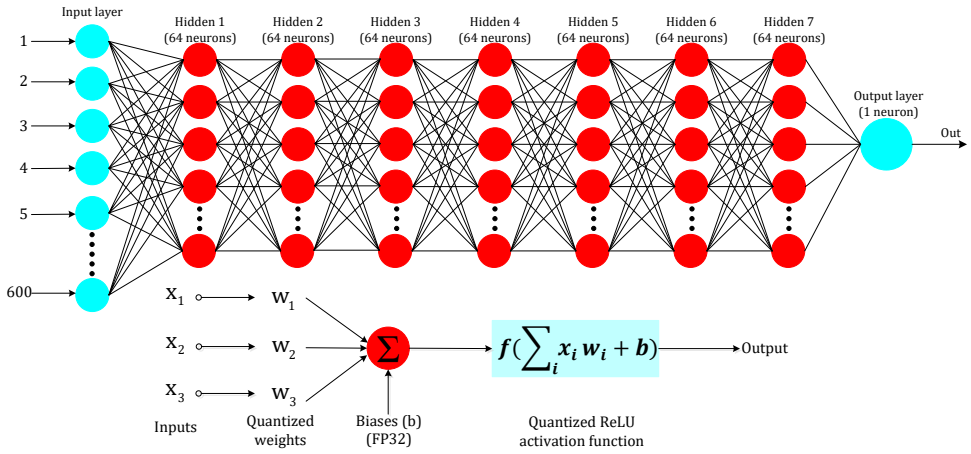


Figure 4.16: A simplified representative multi-layer perceptron

proved to be highly effective in solving numerous problems in the initial stages of ML and DL. However, as the complexity of problems continues to increase in modern applications, there is a pressing need to implement more sophisticated and comprehensive models to achieve practical solutions. The efficacy of these deeper networks in diving complex problems relies heavily on software optimization and hardware accelerators. The complexity of such models often requires substantial memory allocation, which can lead to performance bottlenecks in the supporting hardware. This high demand for memory also raises concerns about power consumption, which is particularly problematic in battery-powered portable devices. These challenges can significantly hinder the practical usability and efficiency of such devices. To address these issues, developers have introduced pruning, pooling, and thinning techniques to reduce the computational load and power requirements of complex NNs [337,338]. Moreover, model quantization effectively

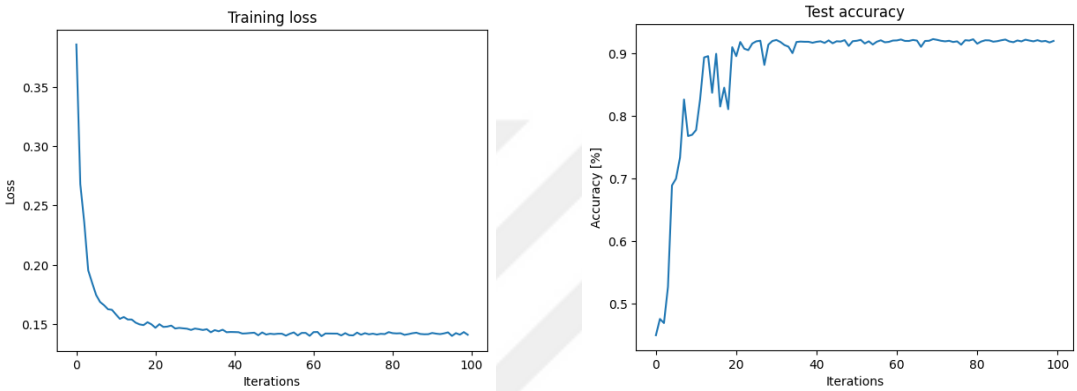
Table 4.5: Resource consumption of Zynq Z720 SoC

Resource	Available	Utilization
LUTs	53200	11591
FFs	106400	14640
BRAM_18K	630KB	61KB
LUTRAM	17400	44
DSP Slices	220	0

balances accuracy with complexity, enabling the integration of NNs into hardware setups. This research confirms that the application of quantization considerably reduces memory requirements and computational complexity. Consequently, quantization emerges as a powerful strategy for minimizing the overhead of implementing sophisticated NNs. Table

Table 4.6: Details about the computer utilized for calculations

Parameters	Value	Unit
CPU Manufacturer	INTEL	-
CPU Variant	i7-7700HQ	-
CPU Clock Frequency	2.8 to 3.8	GHz
CPU Core Size	4	-
Cache Size	6	MB
RAM Size	16	GB
GPU Manufacturer	NVIDIA	-
GPU Chipset	GTX1050	-
GPU RAM Size	4	GB



(a) Cost function (b) Training accuracy

Figure 4.17: Low precision int-2 quantized custom model cost function and test accuracy at 100 epochs: (a) cost function, (b) training accuracy

4.6 details the computer's specifications for training the model, while Table 4.5 outlines the resource consumption on the development board. The model demonstrates efficient resource utilization due to its limited depth. Additionally, the INT2 quantization of our model consumes fewer resources compared to a model operating at full precision. Figure 4.17 offers visual insights into the training and inference phases. In Figure 4.17-b, the variations observed in test accuracy can be attributed to the effects of quantization.

Table 4.7 showcases the model's impressive performance metrics. The recorded accuracy of 92% reflects a high level of precision in predictions. The latency of 5010ns indicates a fast response time, which is crucial for the timely processing of incoming data. Furthermore, a throughput of 200,600 FPS highlights the model's significant processing capacity, allowing it to handle extensive data volumes swiftly. The real-time deployment on the PYNQ-Z1 platform, as illustrated in Figure 4.18, further confirms the system's ability to deliver prompt and reliable outcomes in practical applications. Collectively, these perfor-

mance indicators emphasize the system's proficiency in providing accurate, efficient, and swift processing, making it highly suitable for various demanding applications.

Table 4.7: Performance evaluation of real-time implementation on ZYNQ SoC

Ref.	Architecture	Model	Memory footprint	Batch-size	Epochs	Learning rate	Quant type	Accuracy (%)	Latency (ns)	Throughput (images/s)
Proposed	ZYNQ Z720	MLP	632KB	100	100	0.001	INT2	92	5.010	200600
[331]	ZYNQ Z720	MLP	632KB	1000	100	0.001	INT2	67.4	1.060	943176



Figure 4.18: An overview of real-time implementation on ZYNQ SoC.

4.1.3.3 Conclusion and Discussion In summary, this research has successfully demonstrated the feasibility and effectiveness of implementing a low-precision, QNN model on a ZYNQ SoC. I achieved significant reductions in memory footprint and computational density while maintaining a high accuracy level, facilitated by the use of the Brevitas library for quantization and the FINN framework for execution on the ZYNQ SoC.

This study contributes to the expanding literature on hardware acceleration of low-precision NNs, highlighting their potential to dive into complex problems in various

domains, such as network intrusion detection. The balance between maintaining accuracy and reducing hardware complexity was central to this research. When tested on the UNSW-NB15 dataset provided by the Australian Cyber Security Center, this model achieved an accuracy rate of approximately 92%, which is commendable given the substantial reductions in resource usage.

However, there are limitations to this study, primarily its restriction to a specific dataset, which may affect the generalizability of the results. The decision to utilize 2-bit quantization was context-specific, and different quantization levels might produce varied outcomes.

5 AMD Xilinx Vitis-AI

This section comprehensively examines the five distinct overlay designs explicitly developed for this thesis. Each design has been uniquely tailored to meet the requirements of three separate studies—Study 5.3.1, Study 5.3.2, and Study 5.3.3. I have conducted a series of performance analyses dependent on varying frequencies to assess each design's efficacy. I describe each stage involved in the overlay design process for clarity and educational purposes. Additionally, I specify the software and hardware environments utilized for these designs. The design processes were carried out using PetaLinux 2022.1 and Vivado 2022.1, providing the tools and platforms to develop, test, and optimize the overlays effectively.

5.1 General Framework of A/DAS Multi-Task Application

Figure 5.1 demonstrates how the multi-learning framework aims to enhance task generalization by utilizing insights from diverse tasks. Following the training procedures, which involve the application of publicly available datasets within this framework, I execute the specialized DPU instructions (each uniquely identified and compiled according to the designated DPU-TRD) on the Kria KV260 platform. Subsequent phases detail the methodologies employed for processing and extracting real-time camera data utilizing the DPU instructions. The architecture of these networks integrates diverse functions into a unified task framework, leveraging the complementary relationships among tasks to augment learning efficiency and improve prediction accuracy by utilizing shared features. This research utilizes a shared backbone encoder built on the ResNet architecture because it is accessible and compatible with resource-constrained embedded devices. The framework seamlessly integrates a pre-trained ResNet model, leveraging infrastructure and a multi-task model provided by [339]. This integration contributes to the efficiency of the model's learning process while concurrently diminishing the computational overhead, making it an effective solution for real-world applications. By aligning with the constraints of embedded devices, this approach enhances the feasibility of the study.

I aimed to achieve rapid inference within a multi-task ML framework, emphasizing real-time object detection. To accomplish this, I utilized the Single Shot Multibox Detector

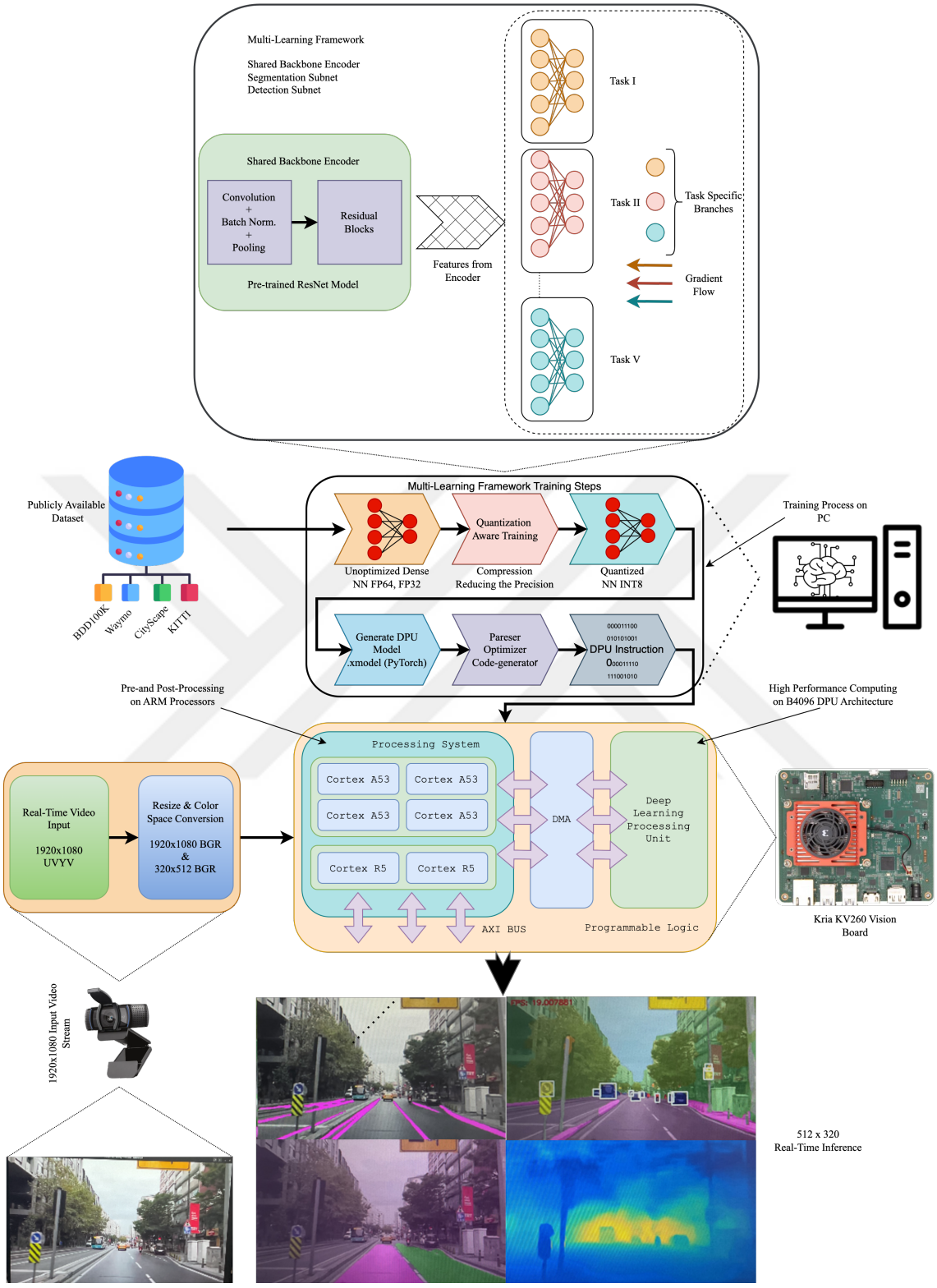


Figure 5.1: General overview of the A/DAS multi-task framework

(SSD) model, which employs a CNN to progressively decrease the feature map's spatial dimensions, effectively reducing resolution. The SSD model excels at detecting and lo-

calizing larger objects by leveraging layers of lower resolution. Moreover, the architecture is enriched with a multi-layer subnet model, an adaptable tool tailored for segmentation tasks. This model is beneficial for estimating drivable areas and depth, making it a valuable asset in real-world scenarios. It includes a learnable upsampling layer, a 3x3 convolutional layer, a batch normalization layer, and activation functions, each playing a crucial role in the robustness and adaptability of the segmentation process.

To enhance the extraction of low-resolution features while preserving greater detail compared to traditional pooling layers, I propose the incorporation of dilated convolution within the convolution block for tasks associated with estimating drivable areas and depth in segmentation. This approach involves an upsampling step to restore spatial and dimensional characteristics lost during downsampling processes. Furthermore, I integrate a ResNet architecture into the decoder segment to refine the semantic boundaries of the data, thereby improving the precision in delineating object boundaries. This integration facilitates a more nuanced understanding and rendering of the scene, which is crucial for accurate segmentation. To prioritize inference accuracy, I emphasize the importance of feature summation, which focuses on aggregating feature maps rather than merely accumulating feature elements. This strategic choice enhances the model's capability to synthesize and utilize higher-level features effectively, thus leading to more accurate predictions and better performance in complex segmentation tasks.

5.1.1 Custom DPU-TRD Design

The DPU-TRD, a sophisticated framework, has been meticulously crafted to facilitate developing and deploying DL models on specialized hardware accelerators such as SoC/MPSoC FPGAs. It incorporates a DPU, an optimized core engineered explicitly for efficiently executing CNNs and other DL algorithms. Designed to enhance DL inference tasks, the DPU-TRD offers a scalable, flexible, and efficient solution, complete with an array of tools, libraries, and documentation that supports the entire workflow from the design to the deployment of DL models on hardware accelerators. This framework enables researchers and engineers to leverage the capabilities of FPGAs and MPSoCs for high-performance, energy-efficient real-time inference across various applications, including autonomous driving, edge computing, and industrial automation. At the heart of DPU-TRD is its

comprehensive support for diverse NN layers and architectures, ensuring seamless integration with popular DL frameworks and facilitating the transition from software-based to hardware-accelerated inference. This feature positions the DPU-TRD as a pivotal element in advancing hardware-accelerated ML, enabling the efficient deployment of sophisticated, computationally intensive models in practical settings.

Within the context of this thesis, I have contributed significantly to the fields of ML and hardware acceleration. I have extensively researched various DPU designs, employing two primary design methodologies: the Vivado flow and the Vitis flow. These designs feature numerous adjustments and optimizations to enhance the execution efficiency of A/DAS tasks. I have focused on developing multiple A/DAS designs, particularly on multi-task learning for A/DAS. The forthcoming sections will delve into the custom DPU design using the PetaLinux 2022.1 and Vivado 2022.1 integrated development environments to create a hardware accelerator overlay. Additionally, this section will demonstrate the performance of the designed DPUs on models provided by AMD Xilinx, showcasing the practical application and efficacy of this work in hardware-accelerated environments.

5.1.1.1 Design Phases This section outlines the programs and interfaces necessary for the Custom Overlay and TRD-DPU design utilized in this thesis. It details the design stages and discusses the required environments: Vivado 2022.1 IDE and PetaLinux 2022.1. The process of designing and integrating a Custom Overlay into the model is also explained, providing a comprehensive understanding of the tools and methodologies employed in developing the hardware architecture.

5.1.1.1.1 Vivado 2022.1 IDE This thesis uses the Vitis Unified Software Platform and Vivado ML 2022.1 IDE for hardware design. Vivado ML, an advanced design suite from AMD, is tailored explicitly for programming adaptive SoCs and FPGAs. It provides a comprehensive set of tools that support various stages of hardware design, including Design Input, Synthesis, Placement and Routing, and Verification/Simulation.

The Vitis platform complements Vivado ML in this integrated design environment, enhancing its utility by facilitating the seamless integration of high-level application code into hardware design workflows. Conveniently, Vivado ML comes bundled with Vitis,

thus requiring only the installation of Vitis. This platform supplies developers with the tools to write applications that leverage the processing system and programmable logic components of AMD's SoCs, optimizing performance and power efficiency. This integration significantly streamlines the process from software development to hardware deployment, proving indispensable for creating sophisticated computing solutions.

To develop embedded system designs with ARM-embedded processors and compute cores on PL, follow the Vitis system design flow shown in Fig 5.2. This structured approach ensures smooth integration of software and hardware components, enabling efficient design and implementation of complex systems. Figure 5.3 illustrates the overlay design follow-

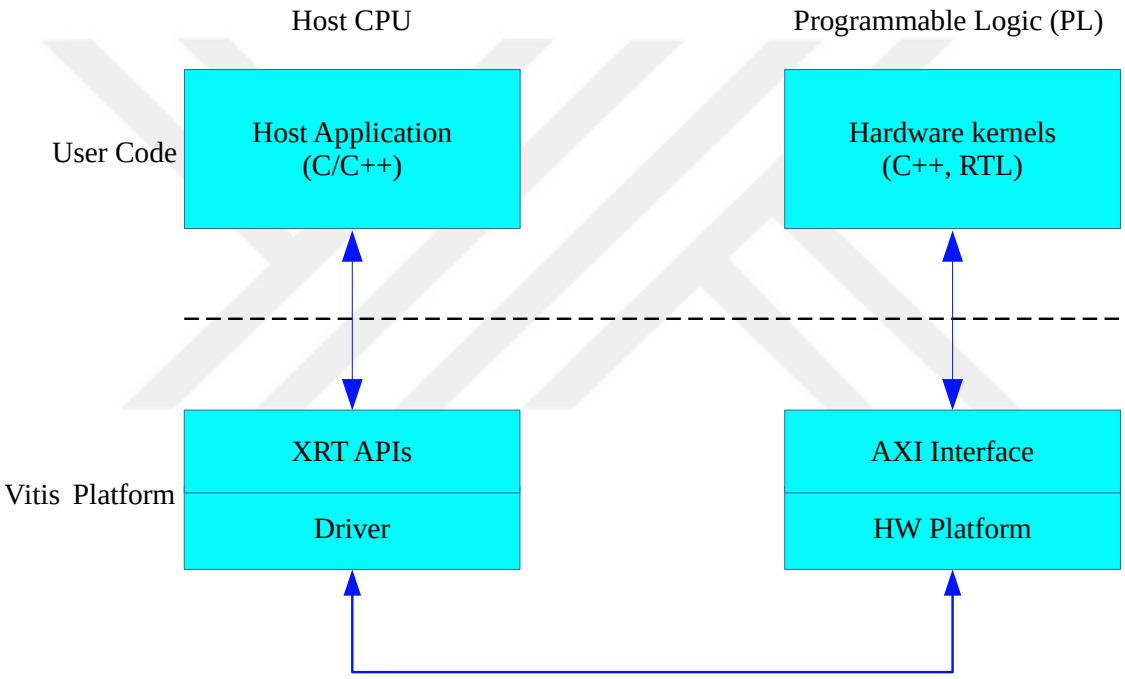


Figure 5.2: Vitis heterogeneous embedded system design flow

ing the execution of the .tcl file in the DPU IP v4.1 (DPUCZDX8G) environment. This hardware architecture integrates key components such as *zynq_ultra_ps_e*, *dpu_trd*, and various auxiliary IPs. Each configuration specified in Table 5.1 was meticulously designed to optimize the *dpu_trd* setup, with distinct SD card images generated for each design iteration using PetaLinux 2022.1. These design efforts aim to achieve optimal performance, necessitating thorough parameterization and careful attention to detail. This is particularly crucial given the challenges posed by the constraints of employing a compact DPU.

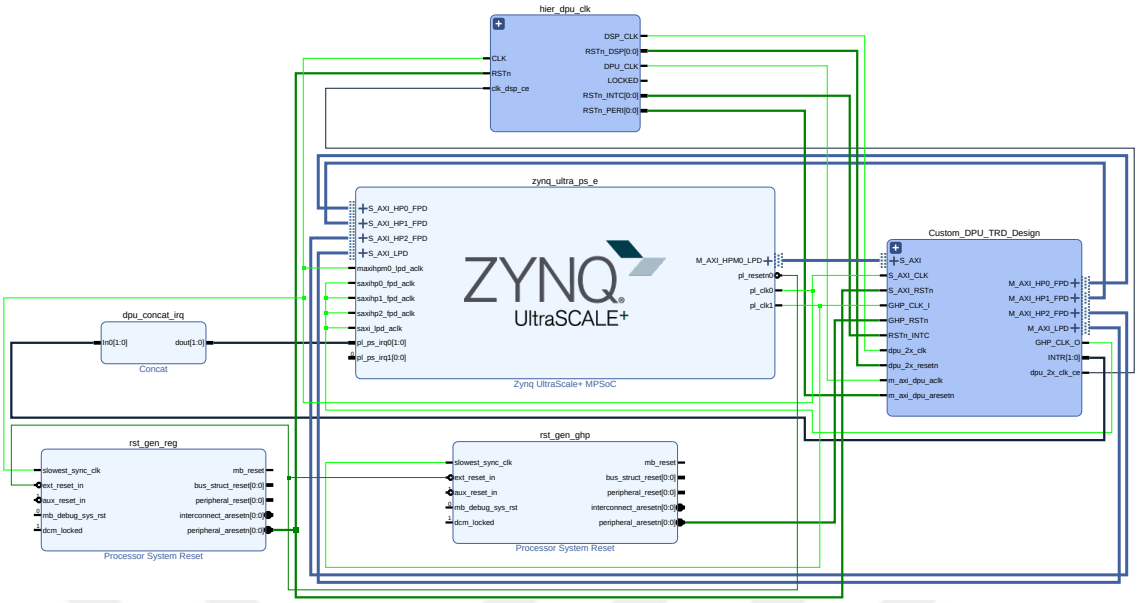


Figure 5.3: Custom DPU-TRD overlay design

5.1.1.1.2 PetaLinux 2022.1 is an embedded Linux SDK designed for FPGA-based MPSoC designs or standalone FPGA projects. It provides developers a customizable environment for configuring, building, and deploying applications targeted at FPGA-equipped hardware. This SDK streamlines the development process for complex hardware systems by integrating a comprehensive suite of tools tailored to the unique needs of FPGA-based architectures. Designing within the PetaLinux environment requires a foundational proficiency in Linux, including the ability to execute Linux commands, alongside an understanding of the OS and host system functions such as the OS version, Linux distribution, security privileges, and elementary concepts of the Yocto. Given these prerequisites, the aspects of this environment mentioned in this thesis will be addressed superficially. An in-depth analysis of these components is beyond the scope of this thesis, as my focus is primarily on the application and performance of specific designs rather than the underlying technical details of the development environment. A Linux or CentOS-based OS is requisite for the installation of PetaLinux. For this thesis, the Ubuntu 20.04 OS, based on Linux, was selected due to its robust features and compatibility. The minimum installation requirements are delineated below. (*Please refer to AMD Xilinx Reference Guide (UG1144) for detailed information*)

- 100 GB free HDD space

- LTS 20.04.2
- 2 GHz CPU (minimum of eight cores)
- 8 GB RAM
- Vitis AI 2.5

After installing, to create a PetaLinux project, we need to download the DPU IP v4.1 (DPUCZDX8G) and development board (Kria KV260) **.bsp** (Board Support Package) files provided by AMD-Xilinx. These files are crucial to ensure compatibility with specific hardware platforms. It is critical to note that the downloaded DPU IP has been configured for ZCU-102's development board. However, for Kria KV260, we will need to make some changes. We need to access the **.tcl** file in the IP and edit the following relevant parts to do this.

```
# set project
#*****************************************************************
dict set dict_prj dict_sys prj_name           {kv260}
dict set dict_prj dict_sys prj_part           {xck26-sfvc784-2LV-c}
dict set dict_prj dict_sys prj_board          {kv260}

#*****************************************************************
# set bd
#   for bd_ooc: None for global, Hierarchical for ooc per IP
#*****************************************************************
dict set dict_prj dict_sys bd_name                    top
dict set dict_prj dict_sys bd_ooc                     None

#*****************************************************************
# set param
#*****************************************************************
dict set dict_prj dict_param  DPU_CLK_MHz             {300}
dict set dict_prj dict_param  REG_CLK_MHz             {100}

#The following parameters correspond to Arch Tab of the IP GUI
dict set dict_prj dict_param  DPU_NUM                 {1}
dict set dict_prj dict_param  DPU_ARCH                {4096}
```

```
dict set dict_prj dict_param  DPU_RAM_USAGE          {high}
dict set dict_prj dict_param  DPU_CHN_AUG_ENA       {1}
dict set dict_prj dict_param  DPU_CONV_RELU_TYPE    {3}
dict set dict_prj dict_param  DPU_ALU_PARALLEL_USER {4}
dict set dict_prj dict_param  DPU_ALU_LEAKYRELU     {0}
dict set dict_prj dict_param  DPU_SFM_NUM           {1}
```

Similarly, we should make the following adjustments based on the design objectives by accessing the **trd_bd.tcl** file in the base folder.

```
dict set dict_prj dict_param DPU_IP_FOLDER          {dpu}
dict set dict_prj dict_param HP_CLK_MHz             {299}
dict set dict_prj dict_param DPU_ALU_LEAKYRELU      {0}
dict set dict_prj dict_param DPU_CONV_WP            {1}
dict set dict_prj dict_param CN_STGY                {flat}
dict set dict_prj dict_param DPU_TIMESTAMP_ENA      {1}
```

After making the necessary edits, open the Vivado IDE and execute the edited **.tcl** file by selecting **'Run Tcl Script'** from the **'Tools'** tab in the window that appears.
After the project is successfully created, follow these steps:

1. Open the project from the directory:

   ```
   /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/Vivado
      /prj\kv260.xpr
   ```

2. Update the Project part to **Kria KV260 Vision AI Starter Kit** and add the CC connection on the project summary page.

3. Access the **Settings -> Bitstream** option from the Flow Navigator window and select the checkbox for the `-bin` file.

4. Click on **Generate Bitstream**.

5. After creating the bitstream, export the hardware by navigating to **File -> Export -> Hardware**, giving a descriptive name, and clicking **Finish**.

6. The resulting bit file will be located at:

```
/home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/Vivado
    /prj\DPU_TRD_300_MHz_wrapper.bit
```

All operations conducted thus far have been aimed at generating the hardware platform. The following steps must be followed carefully to create PetaLinux 2022.1 using the previously downloaded **.bsp** file.

1. Create the PetaLinux project in the specified folder.

```
petalinux-create -t project -s ../home/xilinx/petalinux/2022.1/
    DPU_TRD_300_MHz/xilinx-kv260-starterkit-v2022.1-05140151.bsp
    --name DPU_300_MHz
```

2. Integrate the hardware platform into the PetaLinux project.

```
petalinux-config --get-hw-description=/home/xilinx/petalinux
    /2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/Vivado/prj
```

3. To ensure the proper configuration, please apply the following settings on the configuration screen:

    - Enable FPGA MANAGER
    - Root file-system type "INITRAMFS"
    - Disable "Copy final images to tftpboot"

4. Initiate the process of Kernel Configuration.

```
petalinux-config -c kernel
Device Drivers -->
Misc devices -->
<*> Xilinux Deep learning Processing Unit (DPU) Driver
```

5. Transfer the required recipes to the specified PetaLinux project directory.

```
cp -r /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/
    Vivado/xilinx_zcu102_bsp/project-spec/meta-user/recipes-kernel
    / ./project-spec/meta-user/
```

```
cp -r /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/
   Vivado/xilinx_zcu102_bsp/project-spec/meta-user/recipes-tools/
    ./project-spec/meta-user/
cp -r /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/
   Vivado/xilinx_zcu102_bsp/project-spec/meta-user/recipes-vitis-
   ai/ ./project-spec/meta-user/
cp -r /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPUCZDX8G/prj/
   Vivado/xilinx_zcu102_bsp/project-spec/meta-user/recipes-apps/
   ./project-spec/meta-user/
```

6. Add the following lines to `user-rootfsconfig` in the created project

```
/home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/DPU_300_MHz/project-
   spec/meta-user/conf/user-rootfsconfig ->
CONFIG_vitis-ai-library
CONFIG_vitis-ai-library-dev
CONFIG_vitis-ai-library-dbg
```

7. Update `petalinuxbsp.conf` with the lines provided below.

```
IMAGE_INSTALL:append = " vitis-ai-library "
IMAGE_INSTALL:append = " vitis-ai-library-dev "
IMAGE_INSTALL:append = " dpu-sw-optimize "
IMAGE_INSTALL:append = " resnet50 "
```

8. Execute the root file system (rootfs) configuration. Select the necessary packages, ensuring to exclude the vitis-ai-library-dbg.

```
petalinux-config -c rootfs
```

9. Compile the project.

```
petalinux-build
```

The subsequent step involves the generation of the Device Tree Overlay.

1. Initiate the PetaLinux environment by sourcing it in a separate terminal tab.

2. Initiate the XSCT (Xilinx Software Command-Line Tool).

```
$cd /home/xilinx/petalinux/2022.1/tools/xsct/bin
$./xsct
```

3. Creating a device tree domain and generating the corresponding device tree.

```
xsct% createdts -hw /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/
    DPUCZDX8G/prj/Vivado/prj/DPU_300_MHz_wrapper.xsa -zocl -
    platform-name KV260 -git-branch xlnx_rel_v2022.1 -overlay -
    compile -out /home/xilinx/petalinux/2022.1/DPU_TRD_300_MHz/
    kv260_dt
xsct% exit
```

4. Compile the device tree.

```
dtc -@ -O dtb -o ./kv260.dtbo ./kv260_dt/KV260/psu_cortexa53_0/
    device_tree_domain/bsp/pl.dtsi
```

5. Generate the shell.json file.

```
echo '{ "shell_type" : "XRT_FLAT", "num_slots": "1" }' > shell.
    json
```

6. Now, create a copy of top_wrapper.bin in a different directory and rename it to kv260.bit.bin.

7. Generate the WIC (WIC Image Creator) PetaLinux package.

```
petalinux-package --wic --images-dir images/linux/ --bootfiles "
    ramdisk.cpio.gz.u-boot,boot.scr,Image,system.dtb,system-zynqmp
    -sck-kv-g-revB.dtb" --disk-name "mmcblk1" --wic-extra-args "-c
     gzip"
```

8. At this stage, the following files should be prepared:

```
sdcardimage.wic.gz, kv260.bit.bin, kv260.dtbo, shell.json
```

Finally, we will boot the KV260 using PetaLinux. To accomplish this, please follow the steps outlined below.

1. Utilize Balena Etcher to flash the sdcardimage.wic.gz file onto a 16 GB SD Card.

2. Log in using the username 'petalinux' and set a new password.

3. Configure the Ethernet connection to obtain an IP address for SFTP purposes.

4. Create a directory named TRD_300_MHz in the user area (/home/petalinux) on the development board using an Ethernet connection. Copy the kv260.bit.bin, kv260.dtbo, and shell.json files into this newly created directory. Then, move the TRD_300_MHz directory to /lib/firmware/xilinx/ and enable TRD_300_MHz. The Custom Overlay, created with the assistance of the **xdputil query** command, will be displayed as follows.

```
xilinx-kv260-starterkit-20221:~$ sudo xdputil query
device_core_id=0 device= 0 core = 0 fingerprint = 0
    x101000016010407 batch = 1 full_cu_name=unknown:dpu0
xilinx-kv260-starterkit-20221:~$ sudo xdputil query{
"DPU IP Spec":{
"DPU Core Count":1,
"IP version":"v4.1.0",
"enable softmax":"False"},
"VAI Version":{
"libvart-runner.so":"Xilinx vart-runner Version: 2.5.0-
    c26eae36f034d5a2f9b2a7bfe816b8c43311a4f8  2023-01-22-01:10:05
    ",
"libvitis_ai_library-dpu_task.so":"Xilinx vitis_ai_library
    dpu_task Version: 2.5.0-
    c26eae36f034d5a2f9b2a7bfe816b8c43311a4f8  2022-06-15 07:33:00
    [UTC] ",
"libxir.so":"Xilinx xir Version: xir-
    c26eae36f034d5a2f9b2a7bfe816b8c43311a4f8 2023-01-22-01:08:11",
"target_factory":"target-factory.2.5.0
    c26eae36f034d5a2f9b2a7bfe816b8c43311a4f8"},
"kernels":[{
"DPU Arch":"DPUCZDX8G_ISA1_B4096",
"DPU Frequency (MHz)":300,
"cu_idx":0,
"fingerprint":"0x101000016010407",
```

```
"is_vivado_flow":true,
"name":"DPU Core 0"}]}
```

5.1.1.2 Hardware Configuration Optimizing the usage of hardware resources in design can significantly enhance FPGA performance, leading to higher clock speeds, increased throughput, and reduced power consumption. In the described design, the DPU receives instructions from external memory that direct the computing engine's operations. These instructions are generated by the Vitis-AI compiler, which utilizes sophisticated techniques such as layer fusion and optimization to maintain efficiency and overall performance.

To further enhance the efficiency of the design, I have strategically employed OCM to store input activations, feature maps, and buffer output metadata. This method effectively reduces the demand for external memory bandwidth, maximizing data reuse. The architectural design of the DPU incorporates a computational engine equipped with a deep pipeline that utilizes fine-grained components such as adders, accumulators, and multipliers. By fully leveraging the structural advantages inherent in hardware accelerators, this approach aims to optimize overall performance, thus making significant strides toward achieving superior efficiency in FPGA-based designs.

Given the variation in resource consumption by different convolution architectures within the DPU IP architecture, It is possible to customize these architectures to match the parallelism needs of the convolution unit. Generally, larger architectures consume more PL resources, which correlates with enhanced performance. To further augment the overall efficiency of the DPU, I implemented a Double Data Rate (DDR) approach. This configuration involves allocating a single clock input for general logic operations and a double rate for Digital Signal Processing (DSP) slices.

Evaluating cascade lengths is critical in determining the computational capacity of DSPs. The optimal size of a DSP cascade represents a critical balance, requiring careful consideration between resource utilization and timing performance. Extending the cascade length can decrease resource consumption but may negatively impact timing performance. A sensible strategy, particularly in scenarios involving devices with limited logic resources, involves opting for longer cascade lengths. This approach maximizes the device's capa-

bilities, ensuring a reasonable and efficient utilization of DSP resources. By strategically balancing these elements, the design aims to optimize performance while managing resource constraints effectively.

When DSP element utilization is low, operations focus mainly on multiplications; however, summation and multiplication operations are required when DSP element utilization increases. For the most extensive DPU configuration (B4096), our hardware design emphasizes extensive DSP resource utilization, thereby alleviating the computational intensity of the model. To address the limitations imposed by finite BRAM resources, I have activated URAM memory, enabling the implementation of a more expansive DPU architecture. Through a meticulous process of precise adjustments in allocating DSP, BRAM, and URAM resources, I have optimized the DPU across each design iteration, leaving no stone unturned in our pursuit of the most efficient and effective DPU architecture.

The DPU core has adaptable configuration capabilities, allowing modifications to various DPU properties. For example, I can include Softmax and LeakyReLU activation functions or execute specialized operations like depthwise convolution and channel augmentation. Additionally, I can adjust the DPU's size, ranging from B512 to B4096. This size parameter indicates the maximum number of operations that can be executed per clock cycle and significantly impacts the DPU's parallelism and data processing rate.

Throughout the study, I tested five configurations detailed in Table 5.1. Properly managing BRAM and URAM utilization in these configurations is critical for optimizing performance. BRAM and URAM serve as OCM resources that store weights, biases, and intermediate feature maps, ensuring efficient data access and processing. The DPU architecture has a memory unit with a bit width of ICP*8 (where ICP stands for Input Channel Parallelism) and a depth of 2048. The DPU size variations are reflected in the ICP values, and the memory unit's width is determined by multiplying by 8. Each memory unit accesses one URAM block. If the ICP exceeds 8, at least two URAM blocks are required per memory unit. Typically, BRAM is used for memory units. However, for devices with BRAM and URAM, it becomes necessary to configure the number of URAMs to substitute specific BRAMs with URAMs. The configurations of BRAM and URAM, as outlined in Table 5.1, are established based on the previous criteria, highlighting the critical role of these memory resources in optimizing the DPU's performance. In light of

Table 5.1: DPU configuration cases in studies for energy efficiency through frequency analysis

DPU Configuration (DPUCZDX8G V4.0)	Configuration 1	Configuration 2	**Configuration 3**	Configuration 4	Configuration 5
DPU Architecture	B1152		B4096		B3136
DPU_Clk_MHz	275	300	**275**	300	200
Channel Augmentation	Enabled	Disabled	Enabled		Disable
RAM Usage	High	Low	**Low**	High	
Ultra-RAM per DPU	32	0	64		48
DSP48 Usage	High	Low	High		
Depth-wise Convolution	Enable				
Average Pooling	Enable				
DSP48 Maximal Cascade Length	2	4	4		
Convolution ReLU Type	ReLU + ReLU6		**ReLU + ReLU6**	ReLU + LeakyReLU + ReLU6	ReLU + ReLU6

resource usage and timing performance, the study deduced that the DSP's ideal maximum cascade length is four. The DPU IP core's use of DSP elements forms the basis for whether DSP usage is high or low. In instances where DSP element usage is low, multiplication only is performed, whereas high DSP element usage involves both multiplication and accumulation. By setting DSP usage as high in the hardware design, a reduction in the computational density of the quantized DWSC model led to increased timing performance. Moreover, enabling URAM memory in the hardware design allowed for the use of the more significant DPU architecture, given the device's lack of sufficient BRAM, effectively reducing the resource constraint of the development board. Fine-tuning performed on DSP, BRAM, and URAM culminated in a more optimized PL process, yielding timing requirement values at an optimal level. For instance, the timing summary for a custom overlay design analyzed and synthesized in the Vivado IDE reveals a Worst Negative Slack of 1.135 ns, a Worst Hold Slack of 0.001 ns, a Worst Pulse Width Slack of 2.000 ns, and a Total Negative Slack of 0.009 ns. It is crucial to note that all these values should be non-negative for a design to meet the timing constraints. A negative value in any of these metrics indicates that the design fails to meet the timing constraints, leading to its failure.

5.1.1.3 Software Configuration FPGAs and MP/SoCs are specialized architectures designed to improve CNN applications by enhancing inference rates, reducing latency, and optimizing hardware utilization. In the A/DAS multi-task learning model, the ZYNQ

Ultrascale+ MPSoC architecture employs a co-design approach where the DPU and ARM processors work together. The DPU is dedicated to accelerating the computational workloads of DL inference algorithms commonly used in various computer vision applications. As a result, the DPU performs the CNN computations, while the ARM processor takes care of the pre-and post-processing.

Additionally, DWSC has been adapted to an on-chip pipeline method to enhance parallel processing efficiency and minimize off-chip memory access. This adaptation significantly boosts the run-time scheduling efficiency of the DPU core during layer computations. DWSC reduces the computational power compared to traditional convolution operations, as noted by [340]. The foundational principle of DWSC involves separating the processes into depth-wise convolution (DWC) and point-wise convolution (PWC). The latter employs a standard 1x1 structure following deep convolution processes, as illustrated in Figure 5.4. This structure enables the extraction of feature information from various channels at the exact spatial location, thereby diminishing the computational cost and memory footprint of separable convolutional networks. In an effort to optimize memory utilization on the

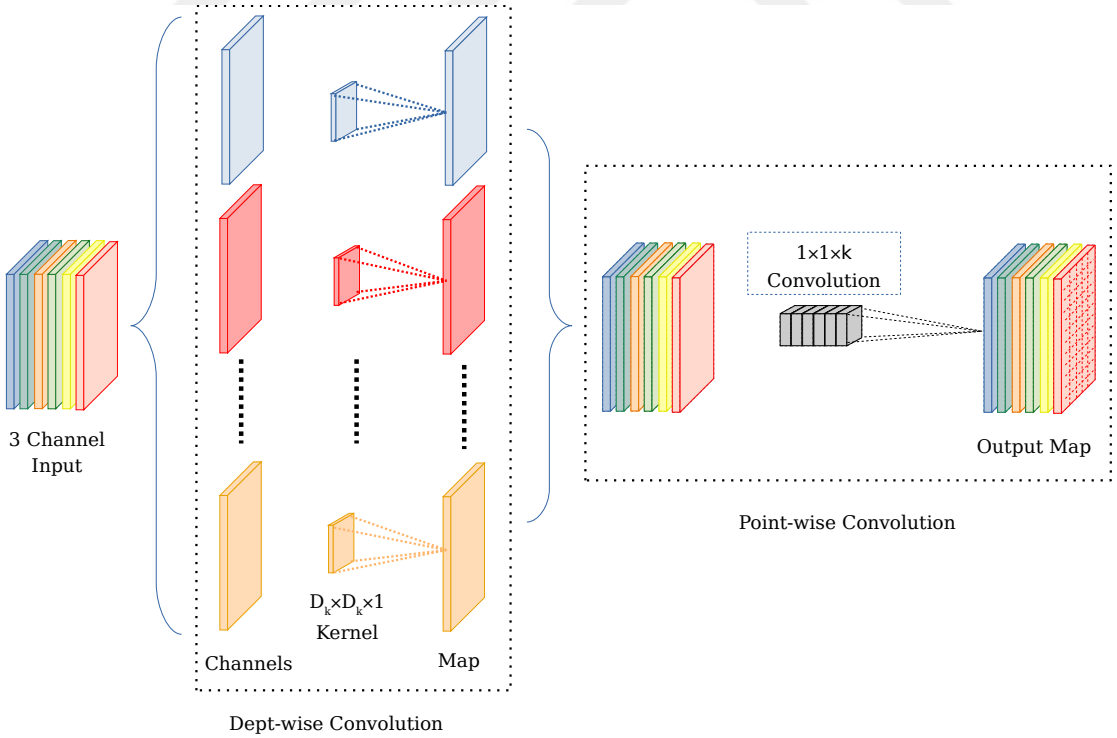


Figure 5.4: Separable dept-wise convolution architecture.

ARM (Quad-Core Cortex A-53) processing engines (PEs), careful data allocation was

enacted to prevent unnecessary re-reading. This facilitated the creation of an adaptable infrastructure for diverse NN models. Similarly, multi-threading and a pipeline architecture were implemented to leverage the DPU and Quad-Core PEs' full potential on the Kria KV260 development board. As a result, high efficiency was attained by mitigating delays during data communication between the PL and the PS. The processing duration of the PL, observable during data transactions via AXI-DMA, underscores this improvement.
A significant software-side enhancement was the quantization of the model, as depicted in Figure 5.5. This modification resulted in a model with a memory footprint well-suited to resource-constrained devices, thereby considerably reducing computational density. Although model quantization may slightly impair inference accuracy, the benefits are sufficiently significant to overlook this minor setback. A detailed account of the improvements and inference results obtained are comprehensively discussed in Section 5.3.

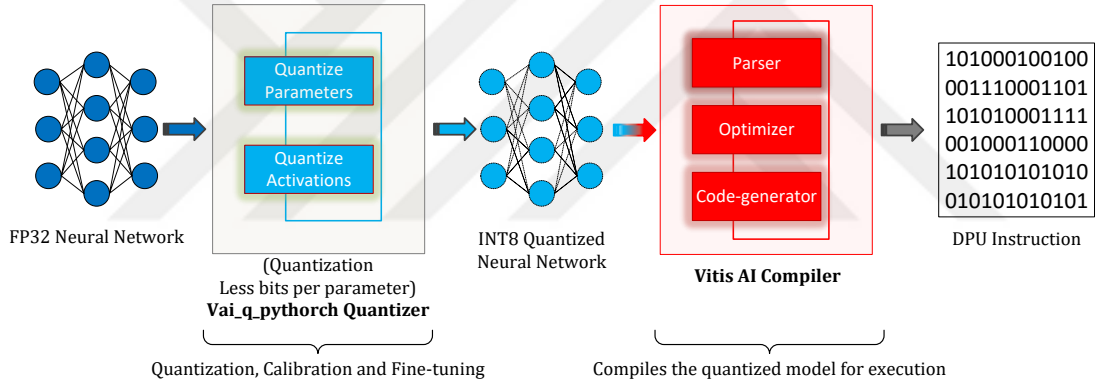


Figure 5.5: An overview of quantization and compilation of fp32 bits NN model

5.1.2 Multi-Learning Model

Multi-task learning networks [139, 341, 342] integrate various functions into a single task, exploiting their interdependencies. In this study, I developed a unified network that combines two learning algorithms for A/DAS, improving the real-time application efficiency. This integration allows for a more accurate function representation through feature exchange, enhancing learning efficiency and prediction accuracy. Multi-task learning also reduces network size and computational complexity by sharing backbone layers, which benefits resource-limited platforms in meeting the demands of real-time A/DAS applications. I have extensively analyzed and detailed the methodology structure, where the

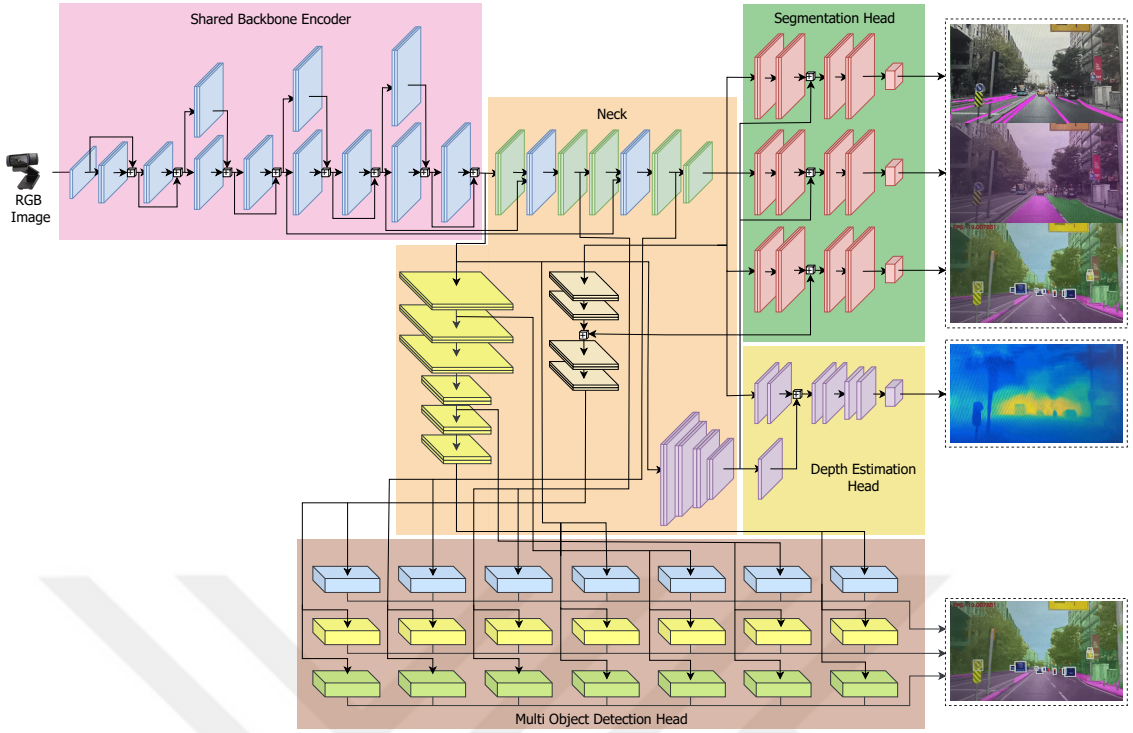


Figure 5.6: An overview of multi-task learning network

shared backbone encoder acts as a feature extractor between tasks. The segmentation head focuses on semantic segmentation, the detection head focuses on object detection, and the depth estimation head focuses on the distance of objects from the camera, demonstrating the network's multi-functional ability. I used Vitis-AI version 2.5 to perform multi-task [339] learning on AMD Xilinx platforms. I followed all necessary steps meticulously to ensure the model's optimal functionality in a Linux environment.

1. Environment requirement

    - Python, PyTorch, OpenCV, ...
    - Vai_q_pytorch (Required for quantization)

```
sklearn, scipy==1.3.1, numpy==1.17.2, tqdm, ninja, six,
    tabulate==0.8.9, graphviz==0.19.1
```

    - XIR Python front-end (Required for quantization)

2. Installation with Docker

    - Obtain the docker image

- Activate PyTorch virtual environment in docker
- Install python dependencies

```
pip install opencv-python==4.5.3.56 imgaug==0.4.0 tqdm
    ==4.60.0 --user
sudo apt-get update && sudo apt-get install cuda-toolkit-11-0
sudo apt-get update && sudo apt-get install cuda-toolkit-11-0
export CUDA_HOME=/usr/local/cuda
sudo update-alternatives --config gcc
pip install pip install mmcv-full==1.2.0 --user
```

The detailed architecture and parameters of the multi-task learning model presented in Figure 5.6 are as follows.

```
==========================================================================
Total params: 15,848,614
Trainable params: 15,848,614
Non-trainable params: 0
Total mult-adds (G): 12.28
==========================================================================
Input size (MB): 1.97
Forward/backward pass size (MB): 368.80
Params size (MB): 63.39
Estimated Total Size (MB): 434.16
==========================================================================
```

5.1.2.1 Shared Backbone Encoder In computer vision, the efficiency of model architectures is crucial, especially for deployment on resource-constrained platforms. The multi-task model from the AMD Xilinx Model Zoo [339] utilizes a shared backbone encoder based on the ResNet-18 architecture, a CNN originally developed for image classification. This model exemplifies how robust feature extraction capabilities can be leveraged across various high-level vision tasks such as object detection, segmentation, and possibly depth estimation, showcasing a practical application of transfer learning principles. This shared backbone architecture is critical for concurrently integrating and processing multiple tasks within a single network framework. It synthesizes and reproduces relevant features across different network modules, enhancing computational

efficiency and reducing the redundancy typically associated with training separate models for each task. The architecture uses region proposal networks for detection, which predict bounding boxes and object classes, and specialized branches for segmentation that delineate object boundaries and classify pixel-level labels for objects and other features like drivable areas or road markings.

I am choosing ResNet-18 as the strategic backbone due to its lower complexity and fewer hyperparameters than deeper networks such as ResNet-101 or DenseNet. This makes it especially suitable for A/DAS, where the demands for real-time processing on embedded devices are severe. The shared backbone approach simplifies the architecture and significantly reduces the computational load, enabling efficient real-time inference without sacrificing the accuracy required for safety-critical applications. The multi-task model, integrated into the Vitis AI framework, is designed to adapt to various AMD hardware platforms, particularly for edge computing scenarios.

Integrating DPU optimizes the execution of intensive convolutional operations, thereby enhancing throughput and latency, which are critical for autonomous systems. Using a shared backbone encoder in the multi-task model represents a sophisticated integration of architectural efficiency and practical deployment considerations. It underscores the potential within the domain of multi-task learning, advancing the capabilities of embedded vision systems in real-world applications.

5.1.2.2 Segmentation Head The segmentation subnet within the multi-task model exhibits a sophisticated architectural design, incorporating several layers optimized for learning fine-grained details essential for precise image segmentation tasks. At the core of this subnet, a series of 3x3 convolutional layers, complemented by batch normalization layers and various activation functions, play a crucial role in maintaining the integrity and non-linearity of feature transformations throughout the network.

The model's architecture is designed and crafted with precision to maintain consistent tensor sizes throughout the convolutional blocks. Any changes in dimensionality are introduced strategically through up-sampling processes, which is a deliberate design choice to ensure that the high-resolution features are preserved and enhanced, thereby increasing the efficiency of the model.

The network employs dilated convolution layers within three distinct convolution blocks to circumvent the common issue of detail loss associated with pooling operations. Dilated convolutions expand the receptive field without reducing the spatial resolution of the feature maps, thereby enabling the extraction of more profound, more comprehensive feature sets without the resolution loss typically associated with standard convolution-pooling sequences.

After the feature extraction phase, up-sampling layers are employed to restore the spatial resolution of the processed images, ensuring that the output segmentation maps are precisely aligned with the original input dimensions. This up-sampling is critical for achieving high-resolution segmentation results for accurate object delineation.

By incorporating skip connections, the model allows for the direct transfer of feature information from the encoder to the decoder segments of the network, enhancing architectural synergy. These connections are instrumental in refining the segmentation output by reintegrating detailed features that might otherwise be lost in deeper layers of the network. This method improves the granularity of the segmentation results and enhances the overall inference accuracy by ensuring that essential spatial details are preserved throughout the network processing pipeline.

During the final stages of the segmentation process, the network does not simply add the individual elements. Instead, it combines and strengthens feature maps with feature summation techniques at specific points. This method significantly improves the model's ability to identify and outline object boundaries with greater precision.

5.1.2.3 Detection Head This part of the multi-learning model employed a SSD as a detection decoder to meet the requirements for real-time applications and facilitate rapid inference. The SSD model utilizes a multi-scale feature map approach for fast object detection. As the CNN architecture progressively reduces the spatial dimensions, the resolution of the feature map correspondingly decreases. This reduction in resolution necessitates using lower-resolution layers for the detection and localization of larger-scale objects, producing the selection of a 4x4 feature map specifically for identifying large objects.

Following the VGG16 layer, the SSD framework incorporates six additional convolutional

layers, with five dedicated explicitly to object detection. This specific layer configuration allowed us to generate six predictions on three distinct layers rather than four, culminating in approximately 8732 predictions. To achieve the specified total of predictions, it is crucial to consider the range of feature map dimensions and the fixed number of default boxes assigned to each feature map location, as depicted in Figure 5.7. Given the SSD

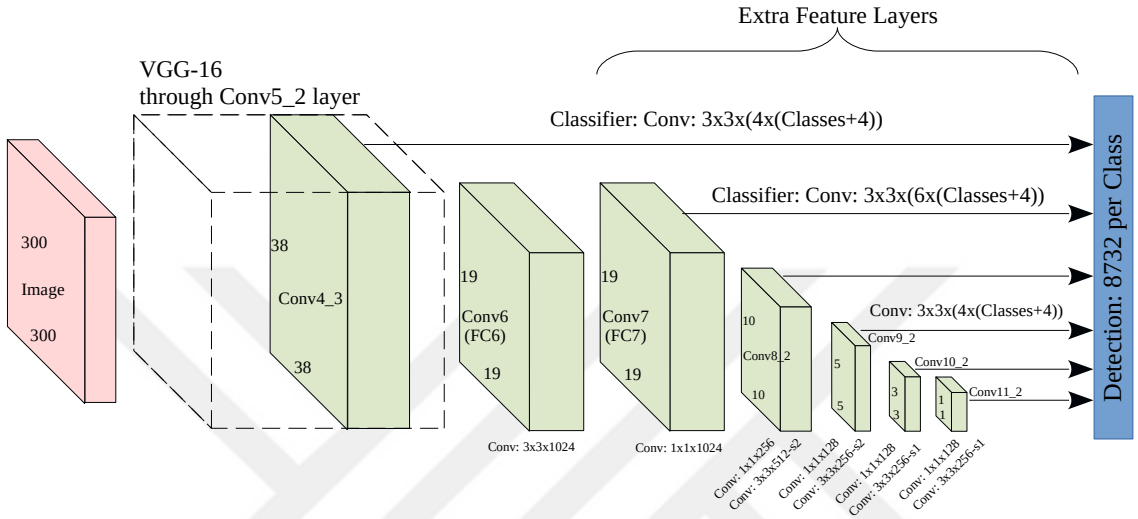


Figure 5.7: SSD single shot multi-box detector [6]

feature maps and default boxes configuration, the calculations for predictions per feature map are outlined as follows:

$$\begin{aligned} Small_maps &: 38 \times 38 \times 4 = 5776, \quad 19 \times 19 \times 6 = 2166, \quad 10 \times 10 \times 6 = 600, \\ Tiny_maps &: 5 \times 5 \times 6 = 150, \quad 3 \times 3 \times 4 = 36, \quad 1 \times 1 \times 4 = 4 \end{aligned}$$

Summing up all predictions results in (42).

$$5776 + 2166 + 600 + 150 + 36 + 4 = 8732 \tag{42}$$

This methodical approach significantly contributes to the multi-task model's efficiency and robustness, enhancing its capability to detect objects precisely under varied conditions.

5.1.2.4 Depth Estimation Head The Depth Estimation Head of the multi-task learning model, integrated with a ResNet-18 backbone, is required for interpreting the spatial arrangement of objects within various scenes, enhancing applications from autonomous

driving to augmented reality. This subnet utilizes a series of convolutional layers, including dilated convolutions, to refine depth predictions while maintaining high spatial resolution. Employing the SILog loss for training, this module emphasizes relative depth errors, which are essential in environments where exact distance measurements are necessary. The architecture's effectiveness is underscored by its ability to deliver precise depth estimations, significantly improving the system's interaction with three-dimensional environments and ensuring robust performance across diverse operational conditions.

5.1.3 Quantization Aware Training (QAT)

Designers have developed more comprehensive architectures to improve the performance of CNN models. Such adaptations, including broader and more profound CNN architecture, have successfully reduced the classification error rate for specific problems. Authors of a particular study [312], utilizing various CNN models, exemplified the relationship between computational density and memory requirements in ImageNet classification. They observed a decrease in the ImageNet classification error rate from 17% to 2.9%. Consequently, expanding the network model incurs an increase in computational complexity. This escalation, in turn, leads to a considerable surge in memory requirements. Additionally, bandwidth concerns arise due to the millions of parameters found in CNN models. Techniques such as model pruning, weight quantization, and activation function quantization [343] aid in reducing computational complexity. However, it should be noted that misuse of the quantization method can lead to decreased inference accuracy. On the other hand, the pruning method can prevent network over-fitting during training. To achieve a goal-oriented model, the designer needs to balance these trade-offs. One potential drawback of quantizing CNN model weights and activation functions is the loss of data, which can be attributed to the inability to restore the floating point after quantization and de-quantization fully. To express this issue in mathematical terms,

$$x = f_d(f_q(x, s_x, z_x), s_x, z_x) + \Delta_x \tag{43}$$

where;

f_d and f_q are de-quantization and quantization functions, respectively. Δ_x is an undeter-

mined small value. Suppose $\Delta_x = 0$, the quantized integer models' inference accuracies are the same as those of the floating point models. Unfortunately, this is not the case. The model performs well after training when the model parameters are in fp32. However, setting the precision to int-8 (8 bits integer) or lower can lead to standard inference even if the network is well-trained. In contrast, the quantized network has a much lower memory requirement than the floating point counterpart, resulting in less energy consumption by the system. As a result, the quantized model is more suitable for battery-powered embedded devices.

This study delves into the implementation of real-time A/DAS for an FPGA-based MPSoC hardware accelerator by quantizing the ResNet18 model with SSD assets. The weights and activation functions of the model were quantified as int-8 bit-width low precision integers and a performance comparison of the network was carried out. The PyTorch framework was utilized to construct the model, and the `vai_q_pytorch` library was used to quantize the weights and activation functions as shown in Figure 5.8. In particular, `vai_q_pytorch` is a library supported by Vitis AI quantizer that operates within the PyTorch framework. Upon successful installation of the required packages in the Linux environment, QAT is initiated using the command `bash ./run_quant.sh multi_task.pth` from the terminal to quantize the multi-task learning model. Provided that the installation and configurations have been executed correctly, the quantization process proceeds as follows.

```
Finished loading model!
[VAIQ_NOTE]: Quant config file is empty, use default quant
    configuration
[VAIQ_NOTE]: Quantization calibration process start up...
[VAIQ_NOTE]: =>Quant Module is in 'cuda'.
[VAIQ_NOTE]: =>Parsing MTNet...
[VAIQ_NOTE]: Start to trace model...
[VAIQ_NOTE]: Finish tracing.
[VAIQ_NOTE]: Processing ops...
213/213 [00:00<00:00, 2532.91it/s, OpInfo: name = return_0, type =
    Return]
[VAIQ_NOTE]: =>Doing weights equalization...
[VAIQ_NOTE]: =>Quantizable module is generated.(quantize_result/MTNet.
    py)
```

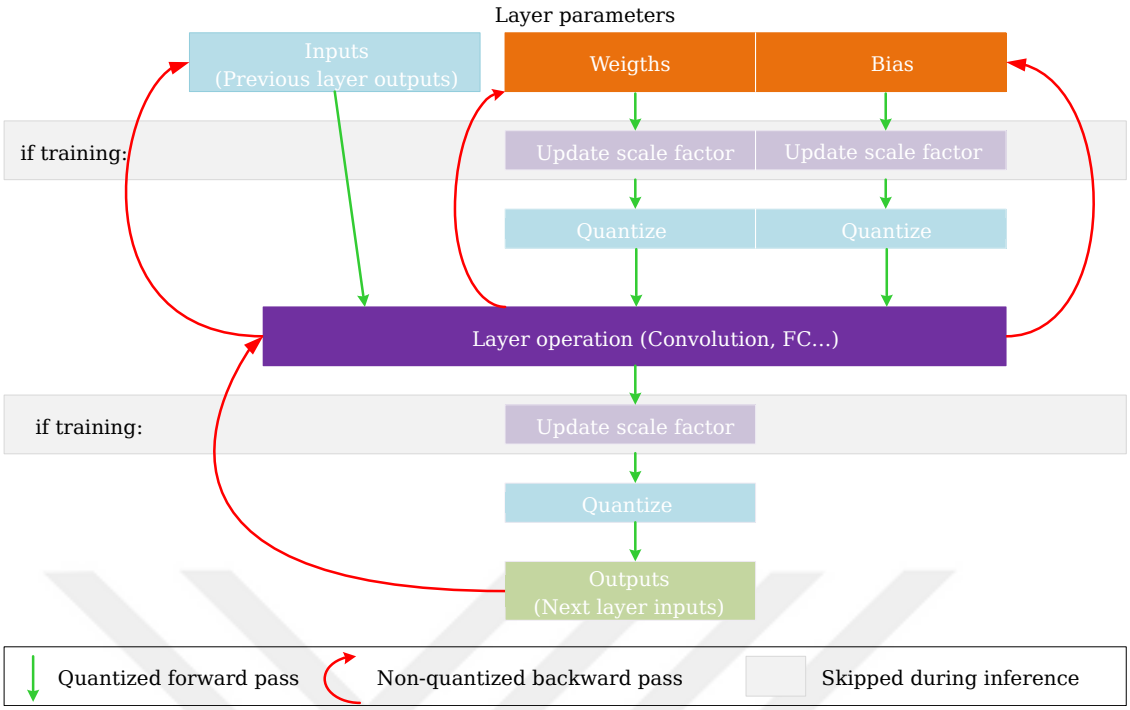


Figure 5.8: Forward and backward pass for QAT process

```
[VAIQ_NOTE]: =>Get module with quantization.
im_detect: 1/10 0.303s
im_detect: 2/10 0.231s
im_detect: 3/10 0.233s
im_detect: 4/10 0.243s
im_detect: 5/10 0.253s
im_detect: 6/10 0.230s
im_detect: 7/10 0.244s
im_detect: 8/10 0.251s
im_detect: 9/10 0.225s
im_detect: 10/10 0.229s
Saveing the detection results...
Writing car VOC results file
Writing sign VOC results file
Writing person VOC results file
[VAIQ_NOTE]: =>Exporting quant config.(quantize_result/quant_info.json)
Conducting quantized Detection test...
[VAIQ_NOTE]: Loading NNDCT kernels...
```

The Vitis AI [344] quantizer takes in the floating point model, conducts pre-processing, and then quantizes the weights and activation/biases at the specified bit-width. The pre-processing performed by Vitis AI folds the batch normalization and eliminates nodes from the model that are not necessary for inference. Thanks to batch normalization, simultaneous learning is possible across layers in the network. Without batch normalization, the use of a high learning rate could lead to the issue of disappearing gradients. However, with batch norms, a higher learning rate can be used since alterations in one layer do not impact the others.

Only the initial value is set as QAT is employed since the learning rate value will undergo automatic updates during training. The authors' extensive investigation and explanation of QAT are presented in their work [345], examining its various components and mechanisms. This article is recommended for those desiring a more complete and in-depth understanding of QAT. The detailed information in this work can provide further illumination and enrichment to the reader's comprehension of this specific area of study.

The QAT technique in NNs desires to minimize the effect of data loss during training, with the model's inference accuracy experiencing only minimal impact. Given that the weight and activation tensors change during NN training, a quantization and de-quantization layer can be added for each varying tensor in QAT [333]. Differing from (43), (44) and (45) can be defined in the following manner:

$$\hat{x} = f_d(f_q(x, s_x, z_x), s_x, z_x) \tag{44}$$

$$\hat{x} = s_x(clip(round(\frac{1}{s_x}x + z_x), \alpha_q, \beta_q) - z_x) \tag{45}$$

Data types for quantized tensors are still floating-point tensors. Therefore, we need to train as if there were no quantization layers. In addition, the main problem with QAT is that such quantization layers cannot be differentiated [333]. On the other hand, the straight-through estimation (STE) [346] derivative strategy excels when used for QAT. The identity function in the clipping range $[\alpha, \beta]$ and the constant function outside of the clipping range $[\alpha, \beta]$ are how STE handles the quantization and de-quantization functions. Thus, the resulting derivatives are 1 if $[\alpha, \beta]$ is in the clipping range and 0 if outside of

the field. We can define symmetric quantization mathematically as in (46).

$$\frac{\partial \hat{x}}{\partial x} = \begin{cases} 1 & if\ \alpha \leq x \leq \beta \\ 0 & else \end{cases} \tag{46}$$

Scaling factors can be discovered during QAT thanks to STE. For instance, the Learned Step-Size Quantization (LSQ) [347] is obtained from the scaling elements' gradient quantization function. Starting from (43), we can get (47) to (50) as follows;

$$\frac{\partial \hat{x}}{\partial s_x} = \frac{\partial s_x}{\partial s_x}\left(\text{clip}\left(\text{round}\left(\frac{1}{s_x}x, \alpha_q, \beta_q\right)\right)\right) + s_x \frac{\partial\left(\text{clip}\left(\text{round}\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right)\right)}{\partial s_x} \tag{47}$$

$$= \text{clip}\left(\text{round}\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right) + s_x \frac{\partial\left(\text{clip}\left(\text{round}\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right)\right)}{\partial s_x} \tag{48}$$

$$\cong \begin{cases} \text{clip}\left(\text{round}\left(\frac{1}{s_x}x\right), \alpha_q, \beta_q\right) + s_x \frac{\partial\left(\frac{1}{s_x}\right)x}{\partial s_x} & if\ \alpha \leq x \leq \beta \\ a_q + s_x \frac{\partial(b_q)}{\partial s_x} & if\ x < \alpha \\ b_q + s_x \frac{\partial(b_q)}{\partial s_x} & if x > \beta \end{cases} \tag{49}$$

$$= \begin{cases} round\left(\frac{x}{s_x}\right) - \frac{x}{s_x} & if\ \alpha \leq x \leq \beta \\ a_q & if\ x < \alpha \\ b_q & if\ x > \beta \end{cases} \tag{50}$$

Here, it is possible to learn or adaptively select different bit-widths for each layer in a model or a uniform bit width for the entire model.

In examining quantized DL models, one must address how layers that perform non-linear transformations, such as ReLU, and those that handle batch normalization are integrated. Traditionally, a straightforward method might involve de-quantizing the inputs to these layers, processing them using standard floating-point operations, and then re-quantizing the outputs. This approach is feasible when only some such layers or specialized techniques for managing these layers in a quantized format are present. Nonetheless, this could considerably slow inference, especially as these layers are typically numerous within DL architectures.

One alternative approach involves carefully dealing with each layer individually. For example, the quantized ReLU can be defined in a way that, although different from the standard textbook definitions, provides a more comprehensive form suitable for demonstrating its operation in a quantized setting. This approach maintains operational efficiency and accuracy in quantized networks at each layer. The development and mathematical formulation of the QuantReLU function are thoroughly described in Section 4.1.3. This discussion is based on concepts initially explored by Lei [333]. For a deeper understanding and a more detailed exploration of these concepts, I recommend referring to the original works by Lei Mao [333], which offer extensive coverage of the experiences and methodologies involved in quantizing NNs.

5.1.4 Datasets

I integrated object detection and segmentation techniques into a multi-learning framework to enhance A/DAS for autonomous and driverless vehicles. For this purpose, I selected four distinct datasets: BDD100K [348], Waymo [349], KITTI [350], and CityScapes [351], each chosen for its unique contribution to A/DAS development. I leveraged the BDD100K and Waymo datasets extensively to simulate a wide range of driving scenarios, mirroring real-world conditions. The KITTI dataset, with its advanced object detection capabilities, was instrumental in refining our framework's detection algorithms. Similarly, the CityScapes dataset, with its detailed semantic segmentation capabilities across nineteen categories, provided a comprehensive analysis of the driving scene. The methodology involved flexibly using these datasets, tailored to each study's specific needs. I documented the application of these datasets, individually or in combination, and presented

this information in this thesis in detailed tables.

The methodology involved merging these datasets to create a robust data foundation. This process significantly improved the accuracy and efficiency of the multi-learning framework. To evaluate our framework's performance, I used Average Precision (mAP) for object detection and Average Intersection over Union (mIoU) for segmentation, as outlined in (51) and (52). I used the mmodified Scale-Invariant Logarithmic (SILog) error metric for depth estimation (53). The findings demonstrated that the approach of combining these datasets led to substantial improvements in these performance metrics, proving the effectiveness of our multi-learning framework in advancing A/DAS technology for autonomous vehicles.

$$mAP = \frac{1}{n} \sum_{k=1}^{k=n} AP_k \tag{51}$$

$$mIoU = \frac{\left(\frac{1}{n_{cl}}\right) \sum_i n_{ii}}{\left(t_i + \sum_j n_{ji} - n_{ii}\right)} \tag{52}$$

Here, n_{ii} denotes true positives, and n_{ji} represents false negatives. t_i signifies the total count of pixels within class i, while n_{cl} denotes the total number of classes considered.

$$\text{SILog Error} = \frac{1}{N} \sum_{i=1}^{N} (\log(y_i) - \log(\hat{y}_i))^2 - \frac{1}{N^2} \left(\sum_{i=1}^{N} (\log(y_i) - \log(\hat{y}_i)) \right)^2 \tag{53}$$

The SILog error measures the logarithmic error between the predicted and true depth values, adjusted for scale.

Where;

- N - Total number of pixels in the dataset.
- y_i - The actual depth value at pixel i.
- $\hat{y}_i$ - The predicted depth value at pixel i.
- $\log(y_i)$ - The natural logarithm of the actual depth value at pixel i.
- $\log(\hat{y}_i)$ - The natural logarithm of the predicted depth value at pixel i.
- The first term $\frac{1}{N} \sum_{i=1}^{N} (\log(y_i) - \log(\hat{y}_i))^2$ computes the mean of the squared logarithmic differences between the actual and predicted values, which reflects the

variance of prediction errors in logarithmic scale.

- The second term $-\frac{1}{N^2}\left(\sum_{i=1}^{N}(\log(y_i)-\log(\hat{y}_i))\right)^2$ subtracts the bias squared, effectively reducing the error metric by the squared mean of the logarithmic differences, addressing systematic biases in the predictions.

5.1.5 Model Training

The careful selection of distinct loss functions for each specific task was necessary due to the complexity of our multi-learning framework. I employed the ResNet model as a shared backbone encoder, pre-trained on the ImageNet [352] dataset. The choice of ImageNet is due to its effectiveness in large-scale image classification, enabling the model to develop significant representations for tasks like object classification and segmentation.
The presented studies underwent about hundreds of thousands iterations to achieve comprehensive learning. During training, the primary focus was generating contextual information or weights from the images. This process continued until the loss functions for each task stabilized at the desired level. QAT transformed the standard fp32 model into a compact int-8 bit-width format. Additionally, I adjusted the input image sizes during training to match our target dimensions. I fine-tuned the entire multi-learning framework by incorporating task-specific labeled data. I updated the parameters of the subnetworks concurrently with the shared backbone network, optimizing the weighted sum of loss functions using various algorithms. To identify the optimal model configuration, I experimented with different hyperparameters, such as learning rate, weight decay, and batch size. Continuous performance evaluation using a validation set ensured model quality.
I employed the `focal_loss2d` classification function for object detection and selected the L_1 loss for bounding box regression. `focal_loss2d` is instrumental in mitigating the impact of class imbalance during training, while the L_1 loss function effectively reduces the influence of outliers in bounding box regression. The model optimization utilized SGD with an adaptive learning rate and a momentum value 0.9.
I set a batch size ratio of 4 in the SSD framework and utilized binary cross-entropy loss (BCELoss) for robust training. I initialized the encoder weights with a pre-trained model tailored for segmentation, employing `LovaszSoftmaxLoss` [353] for pixel-level

classification and semantic segmentation. During training, I chose a batch size of 2 for segmentation and used SGD as the optimizer with a learning rate 1e-3.

Post training and fine-tuning, the multi-learning framework was evaluated on a test set using task-specific metrics: mAP for object detection, Scale-Invariant Logarithmic (SILog) error [354] for depth estimation, and mIoU for segmentation. These evaluations provided critical insights into the model's effectiveness and areas for potential enhancement. The mathematical expressions for SGD, `LovaszSoftmaxLoss`, and SILog are detailed in (54) - (57), respectively.

$$\theta = \theta - \eta.\nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{54}$$

$$\nabla = \frac{\partial}{\partial x}\hat{\imath} + \frac{\partial}{\partial y}\hat{\jmath} + \frac{\partial}{\partial k}\hat{z} \tag{55}$$

∇ represents the gradient operator, η is the learning rate, $x^{(i)}$ corresponds to the training sample, and $y^{(i)}$ denotes the label.

$$loss(f) = \frac{1}{|C|}\sum_{c\in C}\overline{\Delta_{J_c}}(m(c)) \tag{56}$$

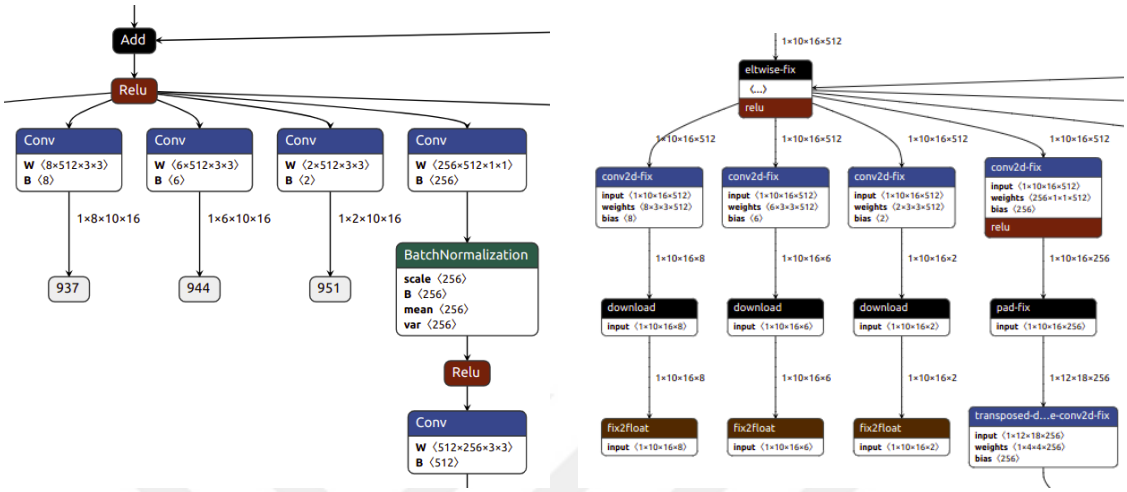
Δ_{J_c} represents the Jaccard loss extension, $m(c)$ denotes the vector of errors, and C signifies the class.

The SILog error measures the logarithmic error between the predicted and true depth values, adjusted for scale invariance. The SILog error can be represented in (57).

$$D(y, y^*) = \frac{1}{2n}\sum_{i=1}^{n}\left(\log y_i - \log {y^*}_i + \alpha(y, y^*)\right)^2 \tag{57}$$

$\alpha(y, y^*) = \frac{1}{n}\sum_i \left(\log(y_i^*) - \log(y_i)\right)$ represents the value of α that minimizes the error for a given value of (y, y^*). For any given prediction y, the factor e^α represents the optimal scaling to align it with the actual ground truth. Since every scalar multiple of y results in the same error magnitude, this demonstrates the property of scale invariance. For enhanced clarity, a segment of the model, which underwent training using QAT with int-8 bit-width and was compiled for execution on the development board, is illustrated in Figure 5.9. The model is presented in both *.xmodel* and *.onnx* formats. File *.xmodel* is designed to be directly imported and run on the specified board, while *.onnx* represents

the more general version. Given the considerable size and intricacy of the multi-learning framework and constrained by page limitations, it is only feasible to include part of the model.



(a) .onnx (b) .xmodel

Figure 5.9: A part of quantized and compiled multi-learning framework

5.2 Energy Efficiency Assessment in Advanced Driver Assistance Systems with Real-Time Image Processing on Custom Xilinx DPUs

This study undertakes a comprehensive examination of the adaptability and performance metrics of the Kria KV260 platform across various configurations, as detailed in Table 5.2 and discussed in Section 5.1.1.2, Table 5.1. The analysis precisely evaluates the characteristics of the DPUs using the system given in Figure 5.10, including frequency, architecture, and resource utilization, such as LUTs, LUTRAM, FFs, BRAMs, URAMs, DSP, and BUFG. The study's findings underscore the Kria KV260 platform's strategic re-

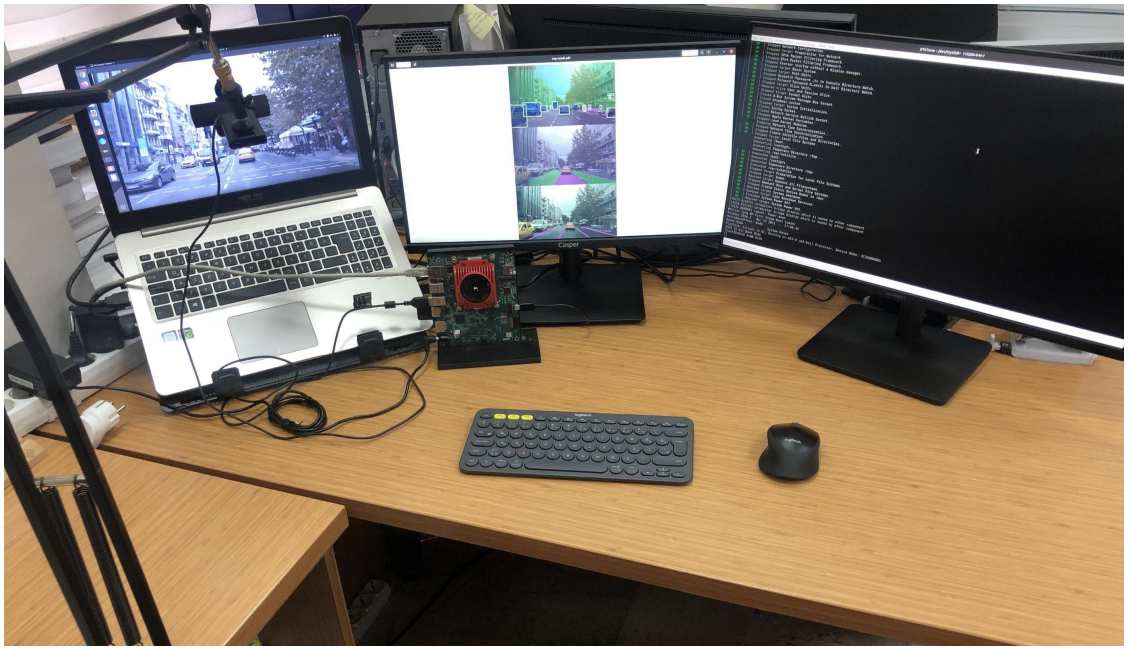


Figure 5.10: Experimental setup

source allocation, a key aspect of its efficiency and adaptability. The platform's utilization rates vary, demonstrating its capacity for customization to meet specific computational demands. DPU frequencies are set at 200 MHz, 275 MHz, and 300 MHz for different configurations, while the architectures vary among B1152, B3136, and B4096. This configuration diversity reflects a tailored approach to balancing computational efficiency and power consumption. For instance, Configuration 4 exemplifies high resource utilization for optimal performance, whereas Configuration 5 represents a balanced compromise, reducing resource usage to enhance efficiency.

The variability in configurations underscores the Kria KV260's versatility, enabling it to

Table 5.2: Comparison of resource utilization of the proposed study in the cases

Kria KV260		Configuration 1		Configuration 2		Configuration 3		Configuration 4		Configuration 5	
Resource	Available	Utilization	Utilization (%)	Utilization	Utilization (%)	Utilization	Utilization (%)	Utilization	Utilization (%)	Utilization	Utilization (%)
LUTs	117120	35955	30.7	35489	30.3	51300	43.8	65329	55.78	46233	39.47
LUTRAM	57600	4068	7.06	3983	6.91	7264	12.61	8041	13.96	5374	9.33
FFs	234240	56595	24.16	57360	24.49	101415	42.3	113127	48.3	83860	35.80
BRAM	144	0	0	121	84.03	67.5	46.88	89.13	79	104	72.22
URAM	64	32	50	0	0	64	100	64	100	48	75
DSP	1248	246	19.71	198	15.87	710	56.89	724	58.01	566	45.35
BUFG	352	4	1.14	4	1.14	4	1.14	4	1.14	4	1.14
DPU Frequency		275 MHz		300 MHz		275 MHz		300 MHz		200 MHz	
DPU Architecture		B1152				B4096				B3136	

cater to a broad spectrum of applications—from high-throughput processing to energy-conscious deployments. This adaptability positions the platform as an ideal solution for various computational tasks across multiple fields. Moreover, the high-performance variants of these designs have been employed as specialized DPU-TRD Overlays in multi-task learning applications for ADAS, as clarified in Section 5.3. This integration highlights the platform's capability to support complex, multi-faceted applications, illustrating its utility in contemporary technological landscapes.

5.2.1 Experimental Results

To assess the outcomes of the performance analysis within this research, we concentrated on three primary metrics: power consumption, resource utilization, and inference efficiency. I conducted multiple analyses of this scenario, encompassing the correlation between power consumption and thread count, the relationship between instantaneous power consumption and thread count, the connection between inference performance and thread count, and the total resource utilization for each configuration examined. As anticipated, the empirical findings indicated that adopting a larger size DPU architecture leads to increased resource utilization. Additionally, the results elucidate that the frequency directly correlates with the inference rate. The research established that an augmentation in frequency concomitantly enhances the extraction rate under analogous conditions. The analytical discussion of the experimental endeavors is systematically organized under respective subheadings.

Table 5.3 shows the percentages of total resource consumption for five different configurations, offering a clear view of efficiency and resource demand. Configurations 1 and 2 show very similar levels of resource consumption at 26.99% and 27.07%, respectively.

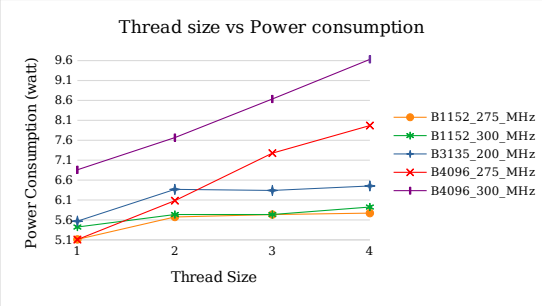
Table 5.3: Total resource consumption for each configuration

Configurations	Total resource consumption (%)
Configuration 1	26.99
Configuration 2	27.07
Configuration 3	44.80
Configuration 4	52.20
Configuration 5	37.94

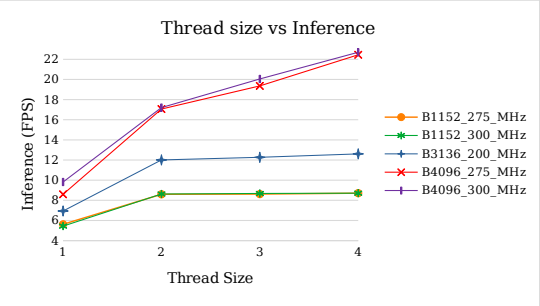
This indicates they are both optimized for resource efficiency or tasks requiring lower computational loads. The proximal values suggest similarities in architectural design or intended operational functionality. Configuration 3 marks a significant increment in resource utilization to 44.80%, reflecting its orientation towards more resource-intensive tasks or a design that inherently demands more excellent computational resources. This jump suggests a custom design for applications requiring higher computational power or memory allocation. Configuration 4, with the highest resource consumption at 52.20%, underscores its allocation towards the most demanding computational tasks, possibly leveraging advanced processing capabilities or heightened parallelism. Lastly, Configuration 5, positioned at 37.94%, represents an intermediate stance in resource utilization, suggesting its design compromises computational performance and resource efficiency, making it suitable for a broad spectrum of tasks with moderate resource demands. The resource consumption profiles collectively underscore the configurations' diverse architectural strategies and application targets, ranging from efficiency-oriented designs to those catering to high-performance requirements.

5.2.1.1 Power Consumption & Inference Efficiency

I analyzed power consumption by employing threading for task allocation and implementing a pipeline architecture on the PL side. The comparative study of DPU configurations, specifically $B4096_275_MHz$ and $B4096_300_MHz$, sheds light on the balance between energy efficiency and computational performance. Figure 5.11-a reveals the $B4096_300_MHz$ setup's higher power consumption across all thread sizes, suggesting it is tailored for superior computational output at a higher energy cost. Conversely, Figure 5.11-b shows that both configurations improve performance with more threads. However, the $B4096_300_MHz$'s advantage diminishes at larger thread sizes, hinting at inefficient energy use for minimal performance gains.

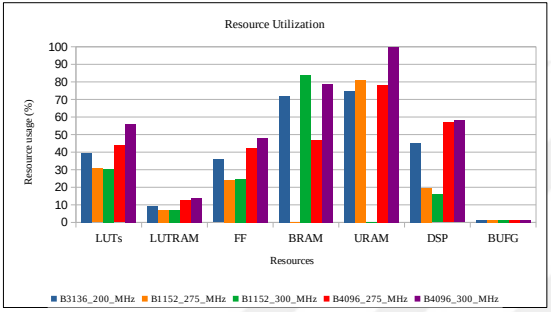


(a) Threads and power consumption relation

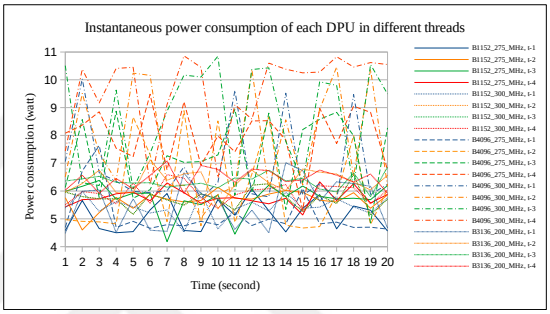


(b) Threads and FPS relation

Figure 5.11: Performance comparison of proposed study based on frequency and DPU architecture



(a) Resource usage for each DPU architecture



(b) Instantaneous power consumption for 20 sec

Figure 5.12: Resource utilization and instantaneous power consumption for each DPU architecture

This analysis emphasizes the need for carefully choosing DPU configurations based on application demands, balancing computational speed against energy usage. It reveals that increases in frequency and power consumption do not always correlate with proportional performance improvements beyond specific thread counts. Consequently, despite the $B4096_300_MHz$ configuration's potential for higher computational output, the $B4096_275_MHz$ offers a more energy-efficient alternative without significantly compromising performance for applications with larger thread sizes. This insight is crucial for optimizing DPU selection, aiming for the best balance between power consumption and computational efficiency in energy-sensitive scenarios.

Further investigation into the $B1152_300_MHz$, $B1152_275_MHz$, and $B3136_200_MHz$ configurations uncovers a similar narrative of prioritizing computational throughput and energy efficiency. The trend indicates higher power consumption at increased thread sizes, particularly for configurations like the $B1152_300_MHz$, which demands more power for

potentially higher speeds. The $B1152_275_MHz$ presents a slightly more power-efficient option with a modest drop in performance, suitable for energy-constrained applications. Meanwhile, the $B3136_200_MHz$ emerges as the most energy-efficient, albeit with lower computational output, ideal for scenarios where moderate inference speeds are sufficient and energy conservation is paramount.

Choosing among the $B1152_300_MHz$, $B1152_275_MHz$, and $B3136_200_MHz$ configurations requires consideration of the application's performance needs versus energy constraints. The $B1152_300_MHz$ is preferable for maximum performance with less concern for energy, while the $B1152_275_MHz$ is a balanced option for those seeking a compromise between efficiency and speed. The $B3136_200_MHz$ caters to applications where energy efficiency outweighs the need for high speed, underlining the importance of aligning DPU selection with an application's computational demands and energy limitations to achieve an optimal mix of efficiency and performance.

5.2.1.2 Resource Utilization In examining the resource utilization and instantaneous power consumption characteristics of various DPU architectures, this analysis unveils distinct operational profiles with significant implications for their application in computational tasks. Figure 5.12-a delineates the utilization percentages of essential hardware resources. Notably, the $B4096_300_MHz$ architecture exhibits pronounced utilization of DSP and BUFG resources, indicative of its optimization for high-speed arithmetic operations key in intensive computational tasks. Conversely, the $B3136_200_MHz$ configuration manifests a more balanced resource allocation, indicating a design oriented towards versatility or generalized computing.

Figure 5.12-b further elaborates on the dynamic power consumption patterns of these DPU architectures over 20 seconds across varying thread sizes (1 to 4). The figure displays a noticeable variability in power consumption, with distinct peaks and troughs that reflect the fluctuating workload demands. Architectures $B1152_275_MHz$ and $B1152_275_MHz$ are characterized by a closer clustering of power usage across all thread sizes, suggesting a commendable level of power management efficiency amidst computational demand oscillations. In contrast, the $B4096_275_MHz$ and $B4096_300_MHz$ configurations reveal broader variances in power consumption, especially at higher thread sizes, underscoring

their deployment in tasks of substantial computational intensity that intermittently necessitate increased power.

The synthesized insights from these observations underscore the critical role of architectural choices in DPU design, particularly emphasizing the balance between resource utilization efficiency and power consumption dynamics. The high resource and power utilization patterns of the $B4096_300_MHz$ configuration aligns it with high-performance applications, whereas the $B3136_200_MHz$'s moderate and balanced profile denotes suitability for general-purpose computing scenarios. This analysis highlights the criticality of aligning DPU architectural selection with the specific computational demands and energy efficiency requirements of the intended application, optimizing performance output and operational sustainability.

5.3 Integration and Optimization of A/DAS Multi-Task Learning Applications on MPSoC-FPGA Platforms through Custom DPU Architectures

This section delves into designing and implementing a custom DPU architecture and multi-task learning networks for A/DAS. I concentrate on integrating hardware accelerators with PL and PS within an MPSoC-FPGA architecture. This approach is essential for effectively executing concurrent A/DAS tasks such as line detection, depth estimation, multiple-object detection, and drivable area segmentation. The development of the custom DPU architecture is intended to improve these A/DAS tasks by utilizing the multi-task learning network's parallel processing capabilities offered by the FPGA. This improvement is facilitated through a hardware overlay design that enables quick prototyping and efficient deployment of various computational tasks. The system's pipeline design ensures a consistent data flow through the pre-processing, core processing, and post-processing stages, thus reducing latency and increasing throughput.

Furthermore, this thesis delves into the challenges and detailed processes of developing a hardware accelerator within the MPSoC-FPGA's PL. This effort includes navigating the complexities of distributing the workload between the FPGA's PS and PL to ensure optimal performance. The FPGA's hardware overlay's adaptability is utilized to dynamically reconfigure different A/DAS tasks, providing a flexible platform for A/DAS applications. Through the architecture presented in Section 5.3.1, 5.3.2 and 5.3.3 I demonstrate the practicality and benefits of employing a custom DPU architecture in a multi-task learning environment. These studies, built on similar principles, algorithms, and hardware acceleration structures, showcase the integrated system's role in enhancing the performance and efficiency of A/DAS functionalities. Each study will systematically outline the specific advancements and improvements, highlighting the integrated system's ability to optimize A/DAS applications on MPSoC-FPGA platforms.

5.3.1 Design and Implementation of a Real-Time Hardware Accelerator on ZYNQ UltraScale+ MPSoC for A/DAS

5.3.1.1 Experimental Setup This section clarifies the experimental configuration's fundamental operational procedures, highlighting the pipeline architecture's paramount significance in facilitating real-time execution for multi-task learning. Subsequently, it will proffer a comprehensive exposition of the requisite data-sets essential for training purposes, followed by an in-depth exploration of the training phase. Lastly, an overview of the intricacies of Hw-Sw co-design shall be presented.

5.3.1.1.1 System setup In this study, I leveraged the unique capabilities of the AMD Xilinx Zynq UltraScale+ MPSoC. This platform, as depicted in Figure 5.13, combines the high computational power of ARM processors with the adaptable PL of an FPGA. I chose this platform for its 4GB of DDR memory, which meets the A/DAS algorithms' memory requirements and enhances system efficiency and reliability.

I exploited the Zynq MPSoC architecture's ability to separate the PS from the PL, which optimized system performance. I processed routine operations in the PS, such as video stream preprocessing, connectivity management, USB interfacing, and system functionalities, using the ARM Cortex A53 processor. This approach allowed us to reduce the video resolution to 320×512, a compromise between image quality and computational load. Allocating standard processing tasks to the PS, I freed the PL for A/DAS's more computationally demanding core functions, such as object detection and image segmentation.

Task allocation strategy was a key factor in enhancing the system's real-time processing capacity, a crucial aspect for A/DAS applications. I significantly improved the system's efficiency by dedicating the PL to performing hardware-accelerated tasks essential for A/DAS, demanding extensive data processing and high parallelism.

Incorporating a multi-threaded pipeline architecture in this system was a strategic design choice that led to significant performance improvements. By enabling parallel data processing, this architecture improved throughput and system responsiveness. It also streamlined data handling between the PS and PL, reducing the system's complexity and resource usage. I effectively harnessed the FPGA's parallelism by aligning FPGA kernels

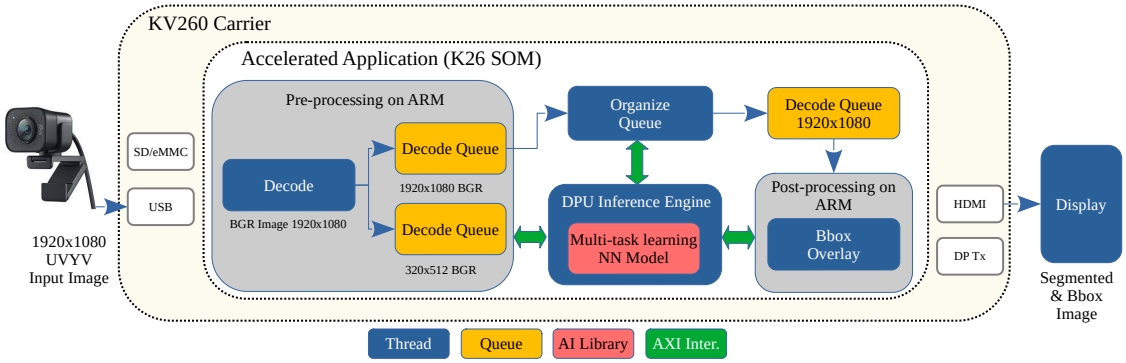


Figure 5.13: General overflow of the multi-task learning model on Kria KV260 with K26-SOM

with specific threads in an organized pipeline, resulting in impressive performance enhancements.

This strategic decision to preprocess the video in the PS instead of the PL resulted from a comprehensive evaluation of the system's capabilities and the A/DAS application's operational demands. This approach ensures efficient resource use and fully exploits the Zynq UltraScale+ MPSoC's strengths. It meets A/DAS's immediate requirements and offers a scalable, adaptable framework for future enhancements. This strategic approach has led to developing an efficient, high-performance system that capitalizes on the integrated PS and PL components for advanced automotive safety solutions. As depicted in Figure 5.13, multiple threads are established as pipelines and operated in parallel to maximize efficiency. This pipeline design strategy yields considerable benefits, contributing to a roughly 50% throughput increase and diminishing design complexity and resource usage, as in Figure 5.14. With each implemented FPGA kernel embodying a single thread, the inherent parallelism within this thread can be fully exploited.

5.3.1.1.2 Datasets I based this research on object detection and semantic segmentation to enhance A/DAS functionality, focusing on driving safety. I utilized open-source datasets like BDD100K, CityScapes, and Waymo, known for their reliability and utility, to support the model's training, testing, and inference phases. Incorporating diverse and densely populated datasets from real-world scenarios is crucial to ensuring the robustness and effectiveness of the work.

I used the BDD100K+Waymo dataset for object detection, which offers various au-

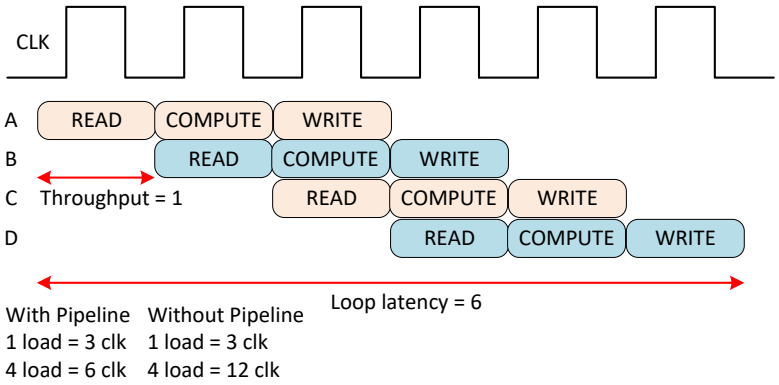


Figure 5.14: Kernel pipeline architecture

tonomous driving scenes. I employed the BDD100K+CityScapes dataset for semantic segmentation, which includes 16 different categorization classes. This study provides a detailed overview of the dataset distribution strategy, as shown in Table 5.4, ensuring a thorough and practical approach to enhancing A/DAS functionality. In assessing the

Table 5.4: Use of data-sets

Datasets	BDD100K+Waymo	BDD100K+CityScapes
Train	70798	72975
Validation	10202	10500
Test	20150	21525

performance of the model, I employed well-established evaluation criteria, specifically the mAP as expressed in (51) for object detection and the mIoU as indicated in (52) to evaluate segmentation quality. The findings underscore the notable improvement in inference accuracy and efficiency achieved by consolidating datasets into shared categories in multi-task learning.

5.3.1.1.3 Model training Given that the multi-task learning model comprises a foundational backbone layer and two associate sub-layers, the training procedure necessitates a meticulous approach, particularly in selecting loss functions. Table 5.4 enumerates the datasets used to train the model. The model's substantial computational demands, attributed to its expansive capacity and the handling of extensive datasets, initiated the utilization of a high-performance GPU. A strategy was employed to mitigate the com-

putational intricacies inherent in end-to-end training wherein the backbone layer and the subnets underwent independent training.

The shared backbone encoder underwent pre-training using the ImageNet dataset, which is well-known for its exceptional performance in large-scale image classification. This pre-training procedure left the backbone encoder with meaningful feature representations suitable for both tasks. The training protocol for the multi-task model extended across multiple stages. Initially, during the training of the segmentation sub-net, the multi-object recognition subnet was intentionally excluded from the training process. Subsequently, a reversal was executed, deactivating the segmentation subnet to exclusively train the multi-object recognition subnet, concluding the training regimen.

Incorporating labeled data during the pre-processing phase initiated the refinement of the multi-task learning model. This refinement enabled us to update the parameters of the shared backbone encoder and their associated subnet parameters. Optimization algorithms were applied to minimize the weighted sum of the loss function. To achieve the most effective configuration for the multi-task learning model, a comprehensive set of experiments was conducted to investigate hyperparameters, encompassing variables such as learning rate, weight updates, and batch size. I relied on the validation test in Table 5.4 for optimal model performance.

While training, I initially defined a learning rate of 1e-4 and a four-batch size. To ensure practical model training, I dynamically adapted the learning rate as required. For optimization, I utilized the SGD algorithm with the `MultiBoxLoss` criterion.

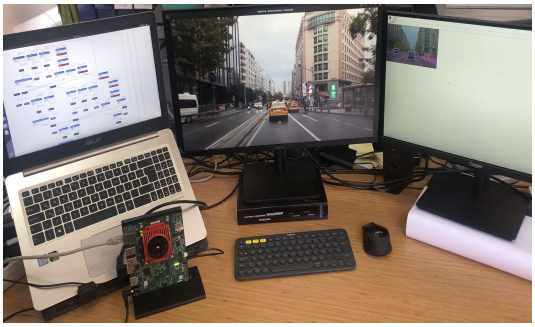
I employed the `LovaszSoftmaxLoss` function and specific parameter configurations in image segmentation. These parameter settings encompassed a weight decay coefficient of 5e-4, a default momentum value of 0.9, and a gamma value of 0.1. I adopted the `LovaszSoftmaxLoss` function because it is suitable for pixel-level classification and semantic segmentation tasks. Furthermore, the model's optimization process relied on the `SGD` optimizer with a learning rate set to 1e-3 and batch size set to four.

5.3.1.2 Results and Discussion This section exclusively presents the results and discussion pertinent to this study, with findings from other related studies delineated in their respective sections. Given the partial overlap in models and platforms utilized across these

studies, a focused comparison was conducted with similar research documented in the literature. Some of the comparative analyses discussed here may also be featured in another of the studies. Figure 5.15-a illustrates the experimental setup, which comprises three principal components: a Logitech StreamCam, the Kria KV260, and a high-resolution FHD display for presenting the results. Additionally, Figure 5.15-b provides a real-time zoomed-in view of the application.

The workflow commences with the real-time video stream being fed via USB, followed by a preprocessing phase within the PS. During this phase, the video's original resolution of 1920×1080 is adjusted to 320×512 to match the model's input requirements. The scaled image is then transmitted to the PL, specifically to the DPU, via the AXI interface. The DPU, a key component in this configuration, undertakes complex computational and convolutional tasks. Following DPU processing, the extracted semantic features experience further refinement within the PS via the AXI interface. Subsequently, these refined features are resized to the desired dimensions and routed to the output, suspended for further analysis and application.

To validate the robustness of this research, I launched an extensive literature review, focusing on similar platforms and related endeavors. This comprehensive review revealed a significant trend: the escalating adoption of MPSoC FPGA-based applications in multi-task learning models. This trend underscores the growing recognition of MPSoC FPGA platforms' potential to substantially augment the capabilities of such models, reflecting their increasing viability and effectiveness in advanced computational tasks.



(a) A thoughtful exploration into the real-time implementation of multi- task learning



(b) Zoomed-in view of the real-time implementation of multi-task learning model

Figure 5.15: Real-time implementation of proposed study

In MPSoC-FPGA, combining software-based and hardware accelerators shows great

promise for delivering significant outcomes. The seamless integration of hardware and software in a coordinated manner markedly enhances system performance. This phenomenon encourages technological advancements underpinning future autonomous and sophisticated driving assistance systems.

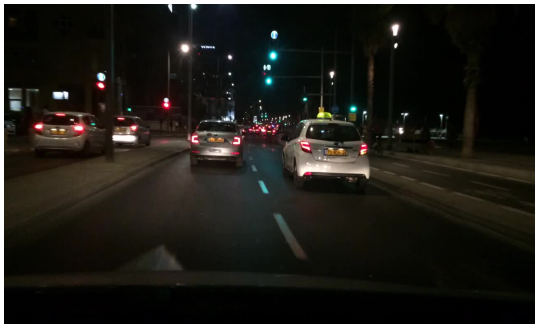
Table 5.5: Performance comparison of the studies in terms of specified metrics

Ref.	Methods	Parameters (Million)	Power (W)	mAP (%)	mIoU / IoU (%)	GOP
[113]	Multi-task learning	N	12.1	N	57.59	13.6
[136]	Multi-task learning	65.2	7.34	N	N	N
[137]	Multiple object det.	0.115	3.118	N	67	N
[139]	Multi-task learning	11.4	7.19	51	56.62	N
[355]	Multi-task learning	N	14.3	62.8	57.6	9
Multi-task learning model (Proposed)	Multi-task learning	21.7	**6.95**	**51.2**	**57.65**	**13.731**

The comparative analysis, as detailed in Table 5.5, shows that the present study achieved superior mAP and mIoU metrics, critical model accuracy, and reliability indicators in segmentation and object detection tasks. Remarkably, this analysis also demonstrated a reduction in power consumption by approximately 4% compared to [139], despite an increase in the parameter count. Upon further examination of Table 5.5, it is evident that this study significantly improved GOP efficiency benchmarks.

Moreover, resource utilization was reduced by 6% relative to prior efforts, underscoring the current research's operational efficiency and performance optimization strides. Furthermore, the study attained a throughput of 47.5 FPS, facilitated by advanced optimization techniques and enhanced pipeline architecture, showcasing its capability to handle multiple real-time tasks across diverse scenarios. These findings underscore this research's practical applicability and substantial contributions within real-world scenarios, highlighting its value to the field. The effectiveness of the system in object detection and segmentation, even under challenging conditions, is further validated by the data presented in Figure 5.16.

In Table 5.6, I have compared the inference accuracy between two model applications—full



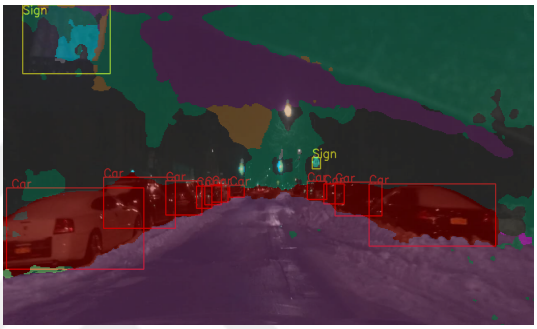
(a) in: summer night time scene



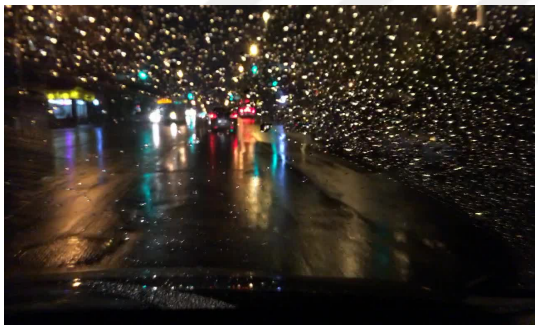
(b) out: segmentation of (a)



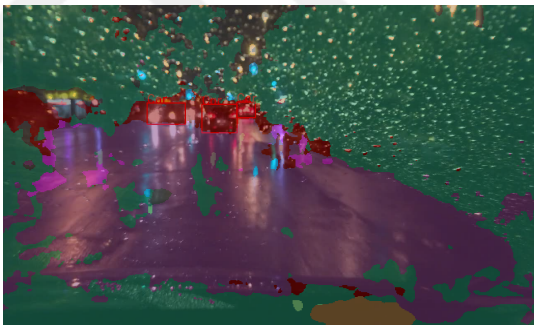
(c) in: winter night time scene



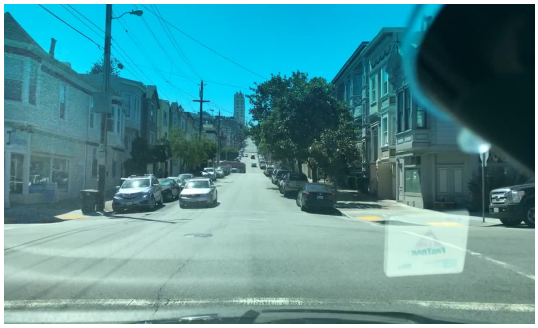
(d) out: segmentation of (c)



(e) in: rainy night time scene



(f) out: segmentation of (e)



(g) in: summer day time scene



(h) out: segmentation of (g)

Figure 5.16: Cognitive analysis of object detection and segmentation performance across varied weather conditions

precision arithmetic (fp32) and low precision arithmetic (int8)—highlighting their computational efficiency and accuracy under varying arithmetic conditions. This comparison aimed to uncover the subtle impacts of precision scaling on overall system performance. The analysis of the results did not reveal any significant differences in performance between the two arithmetic conditions. This finding indicates that low-precision models can achieve inference accuracy comparable to high-precision models, challenging the traditional reliance on high-precision computations for complex tasks. Furthermore, low-

Table 5.6: Performance comparison of the study in fixed point and floating point arithmetic

Multi-Task Learning Model	FP32	int8 bit-width
Multiple object detection mAP (%)	52.86	51.2
Segmentation mIoU (%)	58.16	57.65

precision models offer the distinct advantage of conserving memory resources, critically essential in enhancing the computational efficiency of ML operations. This attribute makes low-precision models a practical option for a wide range of computational tasks, where resource conservation is as crucial as accuracy and speed.

The insights acquired from this analysis are instrumental in guiding the development of more efficient and effective ML models. They demonstrate that precision scaling does not necessarily compromise performance but rather contributes to a more resource-optimized modeling approach. In this study, significant efficiency gains can be achieved by imple-

Table 5.7: Real-time performance comparison of quantized multi-task learning model on different thread size

Thread size	DPU run-time (ms)	ARM (PEs) average run-time (ms)	Performance GOPs	FPS
t-1	19.817	65.541	692.896	25.19
t-2	20.791	71.983	660.447	44.21
t-3	20.857	72.425	658.275	46.3
t-4	21.055	72.474	652.164	47.5

menting multi-threading techniques to enhance performance. As evidenced in Table 5.7, applying multi-threading techniques has led to notable improvements in system performance. However, it must be acknowledged that an increase in the number of threads also results in an extended DPU runtime. This observation underscores a critical trade-off,

necessitating careful consideration and optimization in the design of threads. Determining the optimal thread arrangement is essential to balance computational efficiency and runtime effectiveness, ensuring that the enhancements in performance do not adversely impact the overall system efficiency.

In addition, I have concluded that quantized models demonstrate superior performance compared to fp32 models in terms of resource consumption and computational efficiency. This superiority is demonstrated by maintaining a consistent power consumption profile while preserving memory space. The comparison of resource allocation across multiple runs is presented in Table 5.8. The quantized model's ability to optimize resource use without compromising operational efficiency highlights its potential advantages over traditional fp32 formats. The model utilized in this study exhibits promising potential for future research, surpassing other studies in accuracy and resource utilization. This discovery highlights the significant potential of the quantized model to improve the effectiveness and sustainability of computational tasks across various research domains. This

Table 5.8: The resource consumption of hardware architectures employed in similar studies

Proposed multi-task learning				A/DAS multi-task [139]			Autonomous robot car [355]			Multi-task learning [137]		
Resource	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)
LUTs	117120	63257	54.01	117120	63445	54.17	274080	75000	27.3	70560	39772	54.01
LUTRAM	57600	7304	12.68	57600	7180	12.47	144000	N		28800	3650	12.67
FFs	234240	107757	46	234240	112272	47.93	548160	146000	26.7	141120	59045	41.84
BRAM	144	79	54.86	144	135	93.75	912	280	30.7	216	123	56.94
URAM	64	64	100	64	48	75	0	N		N		
DSP	1248	724	58.01	1248	774	62.02	2520	N		360	211	58.61
BUFG	352	4	1.14	352	6	1.7	N	N		196	3	1.53
Platform	Kria KV260 XCK26 SOM			Kria KV260 XCK26 SOM			ZU9			XCZU3EG		
DPU Architecture	1 x B4096			1 x B4096			2 x B4096			1 x B1152F		
Frequency (MHz)	300			300			600			525		

research, which facilitates the execution of multiple tasks on a single development board, offers a promising prospect for future industries focused on resource-constrained devices. AMD Xilinx's architecture of the B4096 DPU IP has been instrumental in realizing this heightened efficiency within constrained resource constraints. Thanks to the interplay of hardware and software optimizations, this work generally delivers a cost-effective and efficient A/DAS solution suitable for real-world applications, with minimal adaptations required for deployment on diverse platforms.

5.3.2 Real-Time Multi-Task A/DAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture

5.3.2.1 Experimental setup This section details the experimental framework necessary for the real-time operation of multi-task learning in A/DAS. The initial part of the discussion introduces the fundamental workings of the model and emphasizes the significance of an efficient pipeline design, which is essential for processing speed and accuracy. Subsequently, it presents the datasets utilized for the scope of this study. The following part delves into the process adopted for training the model. Concluding this section, an overview of the Hw-Sw co-design integrated within this research, along with a comparison to the approaches detailed in 5.3.1, thereby aiming to provide a thorough understanding of the entire operational process. Furthermore, this study augments the previous work given in 5.3.1 by introducing line detection and drivable area segmentation. The methodology for the remaining steps stays consistent with conventional procedures.

5.3.2.1.1 System setup The experimental setup employed in this study replicates the system configuration used in previous research, as illustrated in Figure 5.10. This setup utilizes the AMD Xilinx Zynq UltraScale+ MPSoC Kria KV260 Vision AI development board equipped with 4GB of DDR memory. The selection of this specific hardware is predicated on its compatibility and demonstrated efficiency in executing the algorithms under investigation.

During this work, some special advancements were made in the development of PL. These advancements include the optimization of on-chip and off-chip memory access techniques, the enhancement of pipeline strategies, and the implementation of hardware acceleration. These developments aimed to improve the overall computational efficiency and reduce latency in the system's processing capabilities, enabling more robust and faster data handling and analysis.

5.3.2.1.2 Datasets In this study, I concentrated on enhancing the A/DAS by integrating line detection and drivable area segmentation into the custom DPU-TRD, using the methodologies discussed in 5.3.1. The model was trained using four publicly available datasets: BDD100K, Waymo, KITTI, and CityScapes. Following the approach

outlined in the 5.3.1 study, I utilized the BDD100K+Waymo dataset, noted for its variety of autonomous driving scenarios, and the KITTI dataset, which provides object detection capabilities across three distinct classes for both detection and segmentation tasks. Additionally, the CityScapes dataset with 16 categories was employed for semantic segmentation. Table 5.9 presents a comprehensive overview of the dataset distribution used for training and inference. The methodology assumed in this research introduced an innovative strategy by merging datasets under common categories. This approach was critical in improving the accuracy and efficiency, marking a crucial advancement in applying A/DAS. The data was portioned for model training, testing, and validation, resulting in highly promising outcomes for multi-task learning. Evaluating the performance, I employed standard mAP (51) and mIoU (52) criteria. The results suggest that consolidating datasets into shared categories improved the inference's accuracy and efficiency in multi-task learning.

Table 5.9: Use of datasets

Datasets	Train	Validation	Test
BDD100K+Waymo	70000	10000	20000
KITTI	7480	-	7517
CityScapes	2975	500	1525

5.3.2.1.3 Model training Model training went through similar training procedures as in the previous study. The initial training phase focused on establishing basic contextual information within the images by setting appropriate initial weights. In this first stage, weight and activation quantization were similarly accelerated using QAT to streamline the model and reduce unnecessary computational overhead. Pre-processing adjustments were also made to directly align the input image dimensions with the network input size required for effective object detection.

Following the initial training, the model experienced an extensive fine-tuning process, demonstrating the precision of our approach. This phase involved task-specific labeled data, facilitating precise parameter updates in the shared backbone and task-specific heads. I strategically used optimization algorithms to minimize a weighted sum of various loss functions tailored to the unique demands of each task within the model.

A careful selection of hyperparameters was made to optimize performance. This included learning rate adjustments, weight reduction settings, and batch size configurations. The effectiveness of these adjustments was observed and evaluated using the validation sets provided in Table 5.9. For the object detection head, a `focal_loss2d` was chosen to eliminate class imbalance, and a smooth L_1 loss was used for bounding box regression to reduce the impact of outliers. The batch size was set as 16, and `BCEWithLogitsLoss` was used as the loss function. For the segmentation, SGD was applied to refine the model with a learning rate of 1e-5, a momentum value of 0.9, and a batch size of 2.

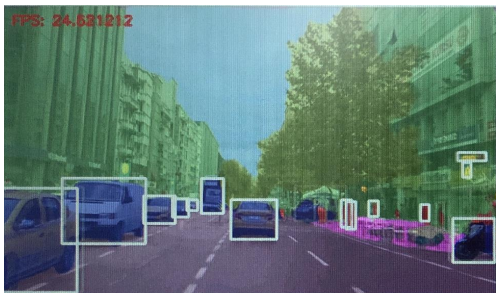
Encoder weights were initialized for segmentation tasks using a pre-trained ImageNet model, increasing the network's capacity to handle pixel-level classifications. To improve the segmentation accuracy, I selected `LovaszSoftmaxLoss`, known for its effectiveness in semantic segmentation. After training and fine-tuning, I evaluated the model using (51) for object detection and (52) for segmentation. These evaluations effectively predicted the model's operational effectiveness and provided baseline information for further improvements.

Table 5.10: Performance comparison of the studies in terms of specified metrics

Ref.	Methods	Params.	Power (W)	mAP (%)	mIoU / IoU (%)	GFLOPs
[113]	Multi-task learning	N	12.1	N	57.59	13.6
[127]	Multiple object det.	0.115	3.118	N	67	N
[136]	Multi-task learning	65.2	7.34	N	N	N
[137]	Multi-task learning	N	14.3	62.8	57.6	9
A/DAS Multi-task Learning	Multi-task learning	11.4M	7.19	51	Segm.: 56.62 Drivable: 81.56 Line: 43.86	25

5.3.2.2 Results and Discussion Performance evaluation was conducted via various A/DAS use cases, including multiple object detection, segmentation, line detection, and drivable area segmentation. The real-time results of these specified A/DAS tasks are demonstrated in Figure 5.17, showing proficient input video data processing.

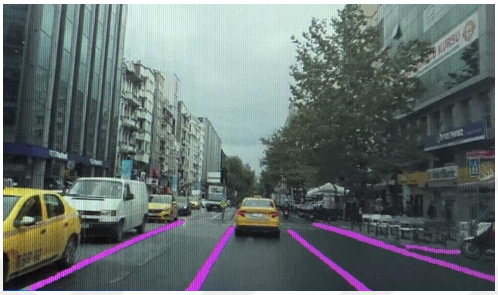
In the previous study, I evaluated the study using the metrics (51) for detection, (52)



(a) Multiple object detection



(b) Drivable area segmentation



(c) Line detection

Figure 5.17: Real-time implementation of A/DAS multi-task learning

for segmentation tasks, and IoU (58) for line detection. I also evaluated performance using the FPS value. Additionally, I emphasized this study's strengths and weaknesses by comparing similar studies in the literature.

$$\text{IoU} = \frac{\text{Area of Intersection}}{\text{Area of Union}} \tag{58}$$

I have analyzed various studies based on specific criteria such as power consumption, number of tasks, and performance GOPs and presented the findings in Table 5.10. My recommended approach outperforms the other studies in these aspects. While the operation in [127] seems to have the lowest power consumption, it only performs a single task of multi-object detection. Researchers performed various object detection and segmentation tasks in a similar study [137]. Although the evaluation metrics are higher than this study's, the power consumption is relatively high. It also needs to catch up on this work in terms of performance. Therefore, as shown in the Table 5.10, this study stands out regarding the number of functions and evaluation results.

The evaluation results for detecting objects using a specific dataset showed an mAP of

51%, indicating its ability to detect objects accurately in real time. Segmenting objects in complex scenarios, it achieved a mIoU of 56.62%, which is impressive. It also showed a good performance in line detection, with an IoU of 43.86%, indicating its ability to detect lines in the environment accurately. The platform's capability in detecting the drivable area existed exceptional, achieving 81.56% mIoU, demonstrating that it can accurately detect the location. Moreover, the study's throughput showed 25.4 FPS at the optimized + pipeline design, indicating its real-time ability to process multiple A/DAS use cases. The fp32 model can provide 25G FLOPs for computational metrics, whereas the int8

Table 5.11: Performance comparison of the study in int8 and FP32 model types

A/DAS Multi-task Learning	FP32	int8 bit-width
Segmentation mIoU (%)	57.95	56.62
Multiple object detection mAP (%)	51.29	51
Drivable Area Detection mIoU (%)	82.63	81.56
Line Detection IoU (%)	43.65	43.86
Performance	25G FLOPs	708.859 GOPs

model can achieve approximately 708,859 GOPs as in Table 5.11. The int8 model has a higher computational rate because it uses reduced numerical precision, which results in less computational complexity and memory usage. As a result, more operations can be performed per second, leading to a higher GOP value for the int8 model. However, there is a slight trade-off between precision and task-specific performance. The fp32 model performs slightly better on segmentation tasks due to its higher numerical precision. On the

Table 5.12: Performance comparison of DPU with and without optimized

Thread size	Optimized + Pipeline			None-Optimized		
	DPU run-time (ms)	ARM (PEs) average run-time (ms)	FPS	DPU run-time (ms)	ARM (PEs) average run-time (ms)	FPS
t - 1	33.551	146.874	10.5	33.901	153.784	8.6
t - 2	34.095	156.342	19	34.194	165.864	15
t - 3	34.788	176.1	22	34.491	186.78	18.86
t - 4	34.839	220.5	25.4	34.766	209.687	21
Memory footprint	62.86 MB			69 MB		

one hand, the fp32 model provides the highest accuracy but requires more computational

resources. On the other hand, the int8 model offers lower accuracy but is more computationally efficient and faster. Ultimately, the choice between fp32 and int8 depends on the application's specific requirements. If the primary concern is achieving high accuracy and no limitations on computational resources, then using fp32 is the best option. However, if the priority is optimizing computational efficiency and speed, the int8 model is a viable alternative with similar performance. Finding the optimal balance between accuracy and efficiency depends on the specific constraints and requirements of the application.

I used multi-threading to improve the efficiency of the work, which resulted in a significant performance increase, as shown in Table 5.12. The AMD Xilinx DPU ID has a maximum limit of four threads. Using multiple threads significantly improves performance—accordingly, the DPU run-time increases, which slightly negatively impacts the extraction speed. Furthermore, I used the pipeline architecture to improve the performance of CNNs processed on the FPGA fabric. Using the Vitis AI analysis tool, I evaluated the performance inference of the platform for all allocation threads and tasks, as shown in Figure 5.18. In this context, each color corresponds to the read and write speeds in megabytes per second (MB/s) for five DDR ports. It also had a low memory footprint,

Table 5.13: Resource consumption of hardware architectures used in similar studies

A/DAS multi-task learning				[137]			[355]		
Resource	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)
LUTs	117120	63445	54.17	70560	39772	56.37	274080	75000	27.3
LUTRAM	57600	7180	12.47	28800	3650	12.67	144000	N	N
FFs	234240	112272	47.93	141120	59045	41.84	548160	146000	26.7
BRAM	144	135	93.75	216	123	56.94	912	280	30.7
URAM	64	48	75	N			0	N	
DSP	1248	774	62.02	360	211	58.61	2520	N	
BUFG	352	6	1.7	196	3	1.53	NA	N	
Platform	XCK26			XCZU3EG			ZU9		
DPU Architecture	1 x B4096			1 x B1152F			2 x B4096		
Frequency (MHz)	300			525			600		

indicating the platform's efficiency in memory usage. As a result, the Kria KV260 Vision AI platform offers high accuracy and efficiency while maintaining low power consumption and memory footprint. Table 5.13 shows the resource usage of similar analyses. In this study, using custom DPU IP, I achieved maximum efficiency of the AMD Xilinx multi-task model at low frequencies and with limited resource usage. The inference performance and resource utilization are commendable compared to the other two works. Hardware

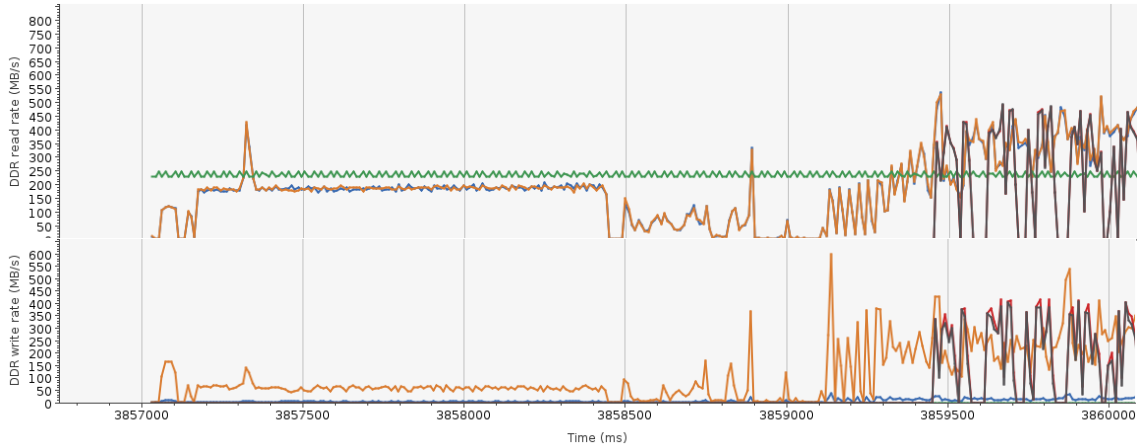


Figure 5.18: Performance evaluation of DDR ports read/write in Vitis AI analyzer

and software optimizations enabled optimal use of the KV 260 board, creating sufficient resource space to include various A/DAS tasks.

5.3.3 Real-Time Multi-Learning Deep Neural Network on an MPSoC-FPGA for Intelligent Vehicles Harnessing Hardware Acceleration with Pipeline

5.3.3.1 Experimental Setup This study constitutes the most intricate research carried out during this dissertation. Although the custom DPU-TRD overlay produced comparable results and was closely aligned with those from other experiments, I incorporated a depth estimation task to conduct a performance analysis in line with projects utilizing similar hardware and models. I successfully executed this task, integrating depth estimation without any degradation in performance. Table 5.14 provides details on the DPU-TRD information I used for this study. To minimize latency in the hardware design, I set the RAM Usage Low and DSP48 High and chose DSP48 Max Cascade Length four as the optimal value.

5.3.3.1.1 System Setup As depicted in Figure 5.19, I carried out the system structure in this study. Distinct from other research, I processed the input video by directly reading it from the SD card. To enhance system performance substantially, I implemented modifications on the PL side, optimizing memory access—a critical factor for efficient data processing and speed. Additionally, consistent with approaches in other studies, I employed multiple threads in pipelines for parallel execution to increase efficiency, as illustrated in Figure 5.13. I aimed to reduce design complexity and computational resource

Table 5.14: DPU configuration for TRD

DPU Configuration DPUCZDX8G	TRD Kria KV260	Model Recompile Required
DPU_Clk_MHz	275	No
Architecture	B4096	Yes
DPU Number	1	No
Channel Augmentation	Enable	Yes
RAM & DSP48 Usage	Low / High	Yes / No
Average Pooling	Enabled	Yes
Depthwise Convolution	Enabled	Yes
Elementhwise multiplication	Disabled	Yes
Convolution ReLU Type	ReLU+LeakyReLU+ReLU6	Yes
DSP48 Maximal Cascade Length	4	No

consumption through these methods, thereby elevating system performance.



Figure 5.19: Experimental setup of proposed study

5.3.3.1.2 Dataset Table 5.15 displays the distribution and utilization of datasets. To increase the accuracy and efficiency of the research, I combined several datasets for different tasks. For instance, I merged the Waymo and BDD100K datasets for multi-object detection. Moreover, I used the BDD100K and CityScapes datasets for segmentation tasks, the BDD100K dataset for drivable area segmentation, and the KITTI dataset for depth

estimation. To evaluate the study's performance, I employed several equations consistent with previous research as in (51) for object detection, (52) for segmentation, (57) for depth estimation, and (58) for lane segmentation.

Table 5.15: Use of dataset

Dataset	Training	Validation	Test
BDD100K+Waymo	70798	10202	20150
KITTI	7480	-	7517
BDD100K+CityScapes	72975	10500	21525

5.3.3.1.3 Model Training In this investigation, I explored the training process of the multi-task learning model, which integrates depth estimation for A/DAS 5.3.2 and employs a newly developed custom DPU-TRD overlay. The significance of the training process has escalated due to the growing number and intricacy of A/DAS tasks. Consequently, I carefully selected appropriate loss functions for each task. I conducted 10,000 iterations of the approximate model to facilitate thorough learning, as delineated in Figure 5.6. This involved acquiring contextual information and weight values from the training images and continuing training until the loss functions stabilized at the desired values.
Additionally, I resized the large-scale training images to conform to the model's input size. Consistent with the previous research, I utilized the ResNet model, pre-trained on ImageNet, as a shared backbone encoder for this study. I used QAT to quantize the fp32 model into a lower precision int-8 bit-width model. I fine-tuned the entire model by incorporating task-specific labeled data and concurrently updated the parameters of the sub-networks along with the shared backbone network. I optimized the combined sum of the loss functions using various algorithms. To identify the optimal model configuration, I experimented with different hyperparameters, such as learning rate, weight decay, and batch size. Moreover, I conducted continuous performance evaluations using the validation sets outlined in Table 5.15.
I employed the `focal_loss2d` classification function for object detection and the L_1 loss for bounding box regression. The `focal_loss2d` proved instrumental in mitigating the effects of class imbalance during the training phase, while the L_1 loss function effectively minimized the influence of outliers in bounding box regression. For the model's optimization, I opted for SGD with an adaptive learning rate and a momentum value 0.9. I

utilized a batch size 16 for SSD and `BCEWithLogitsLoss` for robust training outcomes. Additionally, I initialized the encoder weights using a pre-trained model tailored for segmentation tasks. For pixel-level classification and semantic segmentation, I applied the `LovaszSoftmaxLoss`. SGD was the optimizer for the segmentation tasks with a batch size of 2 and a learning rate of 1e-3.

Following the training and fine-tuning phases, I assessed the model on a test set given table 5.15 using task-specific metrics: (51) for object detection, (52) for segmentation, and (53)for depth estimation. The evaluation process provided essential insights into the model's efficacy and areas needing enhancements. I achieved optimal outcomes through this strict assessment, confirming the model's effectiveness in handling diverse tasks.

Figure 5.20 illustrates the training phase of the study, highlighting the dynamic adjustments in the learning rate throughout each iteration. It also demonstrates how the loss function gradually converges to the optimal value for each task, effectively depicting the process of model optimization and convergence during training.

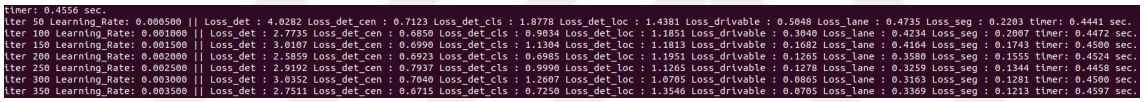
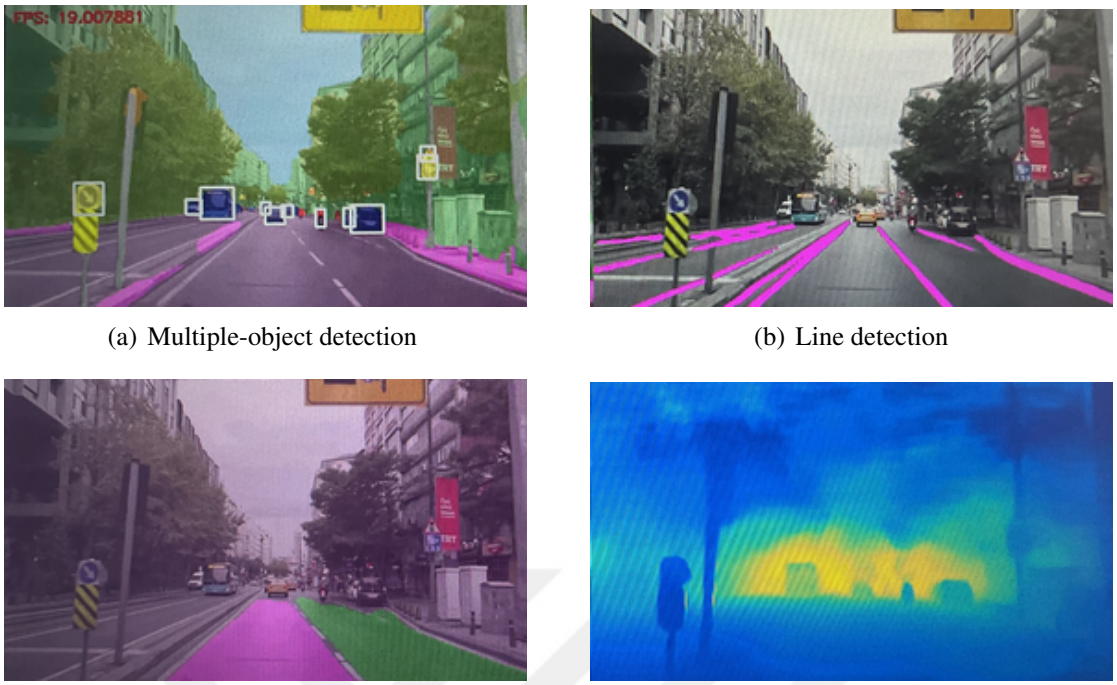


Figure 5.20: Simultaneous training process for each task

5.3.3.2 Results and Performance Evaluation I compared the system's performance when accessing video directly from an SD card versus capturing it through USB. The assessment revealed no significant difference in performance between the two methods. Consequently, I conducted the experiments using video processed directly from the SD card. Figure 5.21 showcases the successful execution of five distinct A/DAS tasks in real-time, highlighting the robustness and efficiency of the approach in simultaneously handling multiple A/DAS functions in practical scenarios.

The effectiveness of this study was thoroughly evaluated by analyzing key metrics relevant to A/DAS, such as power consumption, model parameter size, and GOP. The results were meticulously compared with the most relevant studies in the literature, as detailed in Table 5.16. This research outperformed existing state-of-the-art studies based on these metrics, demonstrating its superiority and innovation. From a hardware perspective, specific refinements, including adjustments to the DPU frequency and strategic management of



(a) Multiple-object detection

(b) Line detection

(c) Drivable area segmentation

(d) Depth estimation

Figure 5.21: Implementation results of proposed multi-learning framework in real-time

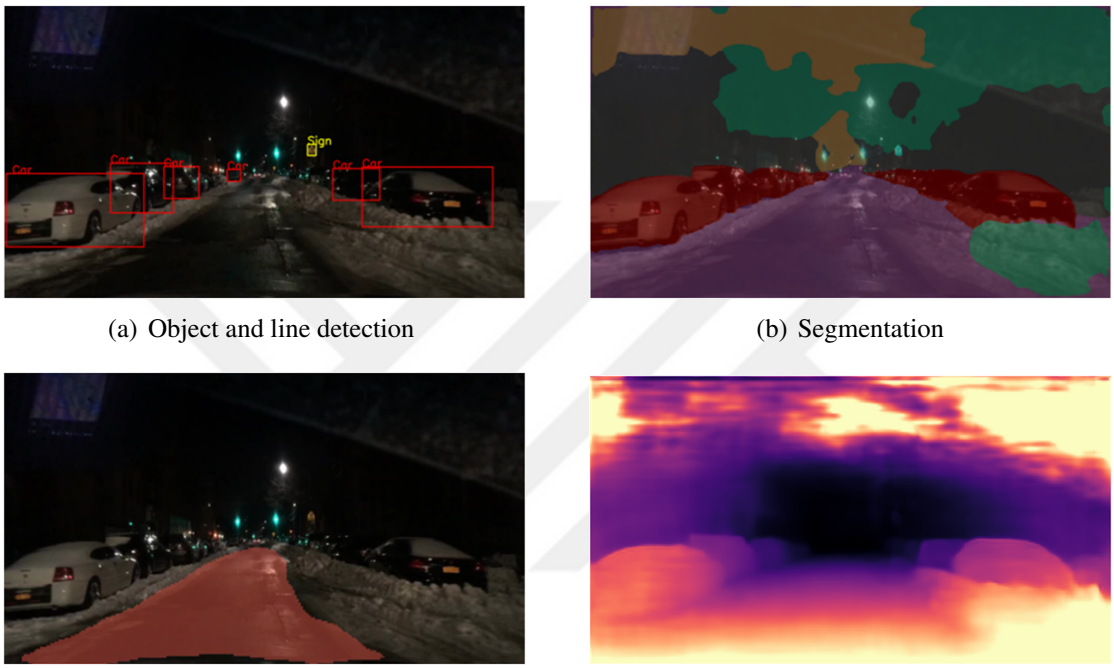
BRAM and URAM, significantly reduced power consumption while enhancing GOP performance. By directly addressing and surpassing the limitations, this research establishes itself as a seminal contribution to the field of A/DAS.

Table 5.16: Compare the performance of the multi-task studies based on the specified metrics

Ref.	Parameters (M)	Average Power Cons. (W)	Evaluation (%)	GOP
[113]	N	12.1	57.59 (mIoU)	13.6
[136]	65.2	7.34	N	N
[137]	N	14.3	Segm.: 57.6 (mIoU) Object: 51 (mAP)	9
[139]	11.4	7.19	Segm.: 56.62 (mIoU) Drivable: 81.56 (mIoU) Line: 43.86 (mIoU) Object: 51 (mAP)	**25**
Multi-learning framework (Proposed)	15.9	**6.920**	Segm.: 57.05 (mIoU) Drivable: 81.63 (mIoU) Line: 43.76 (mIoU) Object: 50.06 (mAP) **Depth Est.**: 9.78 SILog	24.715

To ensure the robustness and long-term viability of the study, I performed comprehensive analyses under challenging weather conditions. As depicted in Figure 5.22 and Figure

5.23, this strict testing process was deliberately focused on the most demanding climatic scenarios. This rendered the presentation of results from milder weather conditions unnecessary. The study's impressive outcomes, demonstrating its compelling performance in adverse weather conditions like snow, rain, and limited night visibility, underscore its resilience and ability to adapt. This ensures safety across diverse and complex environmental conditions. I have compared the results of the previous work, documented in [139], with



(a) Object and line detection

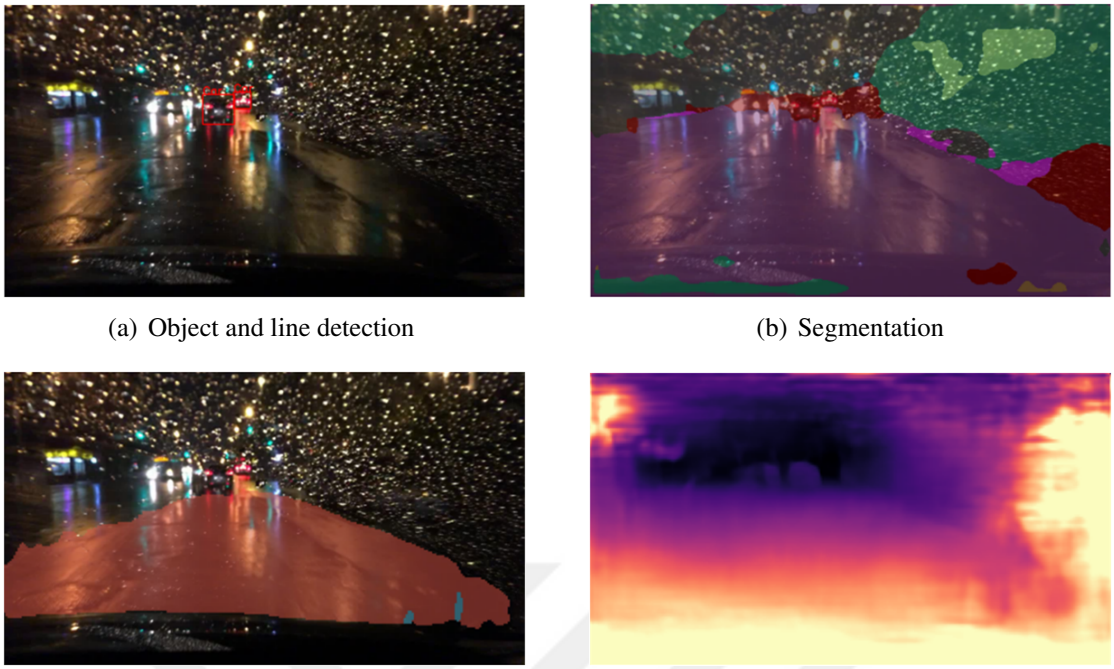
(b) Segmentation

(c) Drivable area segmentation

(d) Depth estimation

Figure 5.22: Performance analysis of the multi-learning framework under snowy night condition.

those of this current research. This analysis specifically focuses on changes in GOPs and FPS. The proposed multi-learning framework shows a slight decrease in GOPs and FPS compared to the earlier efforts. However, it still maintains significant value. This outcome is the decision to set the DPU to operate at a frequency of 275 MHz. In computational science, it is well-understood that the operating frequency directly affects performance. A higher frequency usually enables executing more operations or cycles within a given time frame. However, higher frequencies challenge resource-constrained devices by requiring greater resource demand. Therefore, I have designed the study to include and examine all performance-affecting parameters to ensure a thorough evaluation. Furthermore, as Table 5.18 demonstrates, this study still achieves noteworthy results at lower operating



(a) Object and line detection

(b) Segmentation

(c) Drivable area segmentation

(d) Depth estimation

Figure 5.23: Performance analysis of proposed multi-learning framework under rainy night condition.

frequencies. This study stands out positively when comparing resource usage, as detailed

Table 5.17: Exploring the real-time performance differences of quantized multi-learning framework across various thread sizes

	A/DAS multi-task learning [139]				Multi-learning framework (Proposed)			
Thread size	DPU run-time (ms)	ARM (PEs) average run-time (ms)	Performance GOP/s	FPS	DPU run-time (ms)	ARM (PEs) average run-time (ms)	Performance GOPs	FPS
t-1	33.551	146.874	729.03	10.5	50.470	163.855	490.310	9.026
t-2	34.095	156.342	722.788	19	50.929	174.06	485.281	17.06
t-3	34.788	176.1	716.564	22	50.703	266.341	487.430	19.36
t-4	34.839	220.5	710.514	25.4	50.686	416.639	487.605	22.45

in Table 5.18, particularly when bench-marked against other research in the field. This strategic choice to use a low-frequency DPU-TRD intentionally lowers FPS and GOP figures. However, a detailed review of Table 5.18 shows that the method significantly reduces the consumption of LUTs, BRAMs, DSPs, FFs, and BUFGs compared to similar studies. This method offers a practical solution for resource-constrained hardware implementations and emphasizes the robustness of results for real-world applications. As in other studies, I performed inference comparisons of fp32 and quantized int-8 bit-width models in this study. According to the data presented in Table 5.19, the quantized model demonstrated better results in line detection and memory usage than the fp32 model. However, it showed

Table 5.18: Comparative analysis of resource consumption in studies closely related to our multi-learning framework

Autonomous driving robot car [355]				A/DAS multi-task learning [139]			Multi-learning framework (Proposed)		
Resource	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)	Available	Utilization	Utilization (%)
LUTs	70560	39772	54.01	117120	63445	54.17	117120	51300	**43.8**
LUTRAM	28800	3650	12.67	57600	7180	**12.47**	57600	7264	12.61
FFs	141120	59045	**42.84**	234240	112272	47.93	234240	101415	**42.3**
BRAM	216	123	56.94	144	135	93.95	144	67.5	**46.88**
URAM	N			64	48	**75**	64	50	78.13
DSP	360	211	58.61	1248	774	62.02	1248	710	**56.89**
BUFG	196	3	1.53	352	6	1.7	352	4	**1.14**
Platform	XCZU3EG			Kria KV260			Kria KV260		
DPU Architecture	B1152F			1 x B4096			1 x B4096		
Frequency (MHz)	525			300			**275**		

slightly lower performance in segmentation and depth estimation error. These results emphasize the effectiveness of 8-bit-width quantized models for practical applications, which can provide promising results in real-world situations without significantly compromising performance metrics.

Table 5.19: Performance comparison of the proposed study in int8 and fp32 model types

Multi-Learning Framework	FP32	int8 bith-width
Segmentation mIoU (%)	58.16	56.62
Dept Estimation SILog error (%)	8.96	9.78
Multiple object detection mAP (%)	51.2	50.06
Drivable Area Detection mIoU (%)	82.57	81.63
Line Detection IoU (%)	43.69	43.76
Memory footprint (MB)	69	63.39

6 Conclusion & Discussion

This dissertation focuses on the advancement of A/DAS for autonomous vehicles. The investigation commenced with an exhaustive literature review to identify the existing gaps within the body of knowledge and outline the requisite actions for advancement. To mitigate these deficiencies in A/DAS-related literature, I conducted a detailed examination of small-scale and extensive studies addressing various aspects of hardware design and software optimizations. These discussions are systematically presented in Sections 4 and 5. In the course of this research, specific hardware platforms were selected based on their relevance to the objectives of the study: the PYNQ-Z1 and Zybo Z7 platforms were employed for the experiments described in Section 4, while the Kria KV260 Vision development board was utilized for the endeavors detailed in Section 5. Furthermore, I utilized publicly available software frameworks and supplementary libraries to create the software aspect. Predominantly, the Python Productivity for PYNQ, FINN Experimental Framework, Jupyter Notebook, and associated libraries were chosen for the tasks in Section 4. Conversely, for Section 5, I employed the advanced AMD Xilinx Vitis-AI tools, PyTorch, and related libraries. The rationale and motivations underpinning the selection of these development boards and software frameworks are elaborated in Section 1.2, providing insights into their strategic relevance to the research goals. Furthermore, the findings from these studies, discussions, and outlines of future research directions are thoroughly presented in separate sections corresponding to 4 and 5. These segments provide a comprehensive summary of the findings and offer insight into future research directions to improve A/DAS technologies.

6.1 Detailed Insights from the Studies

Study 4.1.1 effectively demonstrated the superior performance of FPGA-based SoC systems, particularly the ARM+FPGA-based SoC, in executing real-time video processing tasks such as edge detection through Laplacian high-pass filtering. The ARM+FPGA-based SoC significantly outperformed the standalone ARM processor and a high-end multi-core CPU in terms of processing speed and showcased the highest price-performance efficiency. These results underscore the potential of FPGA-based systems in handling

complex, data-intensive video processing applications more efficiently than traditional CPU-based systems. Integrating hardware accelerators like FPGA with conventional processing units (ARM, CPUs) can significantly enhance computational performance while maintaining cost-effectiveness, particularly in fields requiring real-time processing capabilities such as surveillance, medical imaging, and traffic management.

Study 4.1.2 focused on implementing a low-pass FIR filter across various hardware platforms, particularly emphasizing the integration of FPGA and ARM processors within a SoC architecture. The FPGA+ARM-based SoC outperformed a standalone ARM processor by 8.86 times in speed. It was more cost-effective than an Intel i7 setup, making it ideal for cost-sensitive and power-efficient applications such as embedded systems and portable devices. Despite its slower performance than the i7 processor, the FPGA+ARM SoC offers a balance of computational capability, cost, energy efficiency, and robust support for complex signal processing tasks essential in digital communication and multimedia.

Study 4.1.3 has successfully demonstrated the implementation of a low-precision, quantized custom NN model on a ZYNQ SoC, leveraging the FINN framework and PYNQ environment. By quantizing the weight and activation functions, we effectively reduced the model's memory footprint and computational density without significantly impacting its accuracy. We achieved an impressive 92% accuracy rate on the UNSW-NB15 dataset the ACCS generated. The application of the Brevitas library for QAT played a crucial role in maintaining the model's performance during inference by minimizing data loss. This approach not only streamlined the computational process by reducing the precision required but also optimized the memory usage, making it ideal for deployment in resource-constrained environments. This research underscores the immense potential of FPGA-based implementations of NNs using low-precision arithmetic. This approach has the power to effectively address the challenges of high computational demands and large memory requirements in modern AI applications. It opens up promising avenues for advancing the hardware acceleration of NNs, particularly in real-time applications where speed and efficiency are paramount, instilling a sense of optimism for the future of AI hardware acceleration.

Study 5.2 extensively analyzes the optimization of energy efficiency in configurable DPU architectures through frequency analysis on the ZYNQ UltraScale + platform. The study

highlights the potential for system optimization by thoroughly examining various DPU configurations. It underscores the importance of refining these configurations to improve energy efficiency without compromising performance. Findings underscore the critical balance between power consumption, processing speed, and resource utilization, revealing that increased frequencies and thread sizes, while elevating energy consumption, enhance inference performance. Configuration 4 was identified as providing the best balance between performance and resource efficiency despite its higher energy demands. Future energy-efficient and high-performance computing research should focus on advancing machine ML and exploring adaptive frequency scaling techniques. The insights gained from this research will contribute significantly to the fields above, underlining the importance of continued innovation and exploration in this area. The potential for significant advancements in energy-efficient and high-performance computing solutions further underscores the urgency and necessity of this research.

Study 5.3.1 presented an effective solution for managing multitask A/DAS functions on an MPSoC-FPGA platform, significantly reducing memory usage and power consumption. This method improved performance on several essential A/DAS functions, including segmentation and multiple object detection. Using multiple models and a unified learning algorithm within a single hardware framework significantly enhanced accuracy and operational efficiency, making this approach practical for real-time embedded system applications. The experimental results demonstrated that this study achieved better performance and power efficiency than similar studies. Additionally, using a quantized model allowed the study to reach high GOP values while maintaining a frame rate of 47.5 FPS.

Study 5.3.2 introduced a novel and innovative multi-task A/DAS learning approach on an MPSoC-FPGA, with a focus on enhancing tasks such as multiple object detection, lane detection, drivable area segmentation, and semantic segmentation, all while minimizing power consumption. The method incorporates unique software and hardware enhancements, including the integration of multiple models and a unified learning algorithm, which have collectively led to a significant 9% reduction in memory usage. Furthermore, hardware optimizations such as parallelization and pipeline architecture have not only improved efficiency, compatibility, and speed but have also enhanced overall system performance while reducing energy consumption. Employing a single B4096 DPU

has significantly reduced resource consumption, achieving an energy consumption rate of 7.19 watts, a frame rate of 25.4 FPS, and enhanced detection rates across various metrics. Moreover, this system's low power consumption, cost-effectiveness, and compact design make it an attractive solution for real-world applications. By leveraging PL and PEs, the study highlights the feasibility of creating custom hardware accelerators like DPUs for fast and low-latency processing.

Study 5.3.3 similarly emphasized the crucial role of advanced computational resources and algorithms in enhancing automotive safety and convenience through A/DAS. This research distinguished itself by incorporating depth estimation, which, along with a substantial reduction in power consumption, led to enhanced performance in response to increased A/DAS tasks. The employment of MPSoC-FPGA architectures was pivotal, providing parallel computing and architectural flexibility that significantly boosted performance. Further, this study improved the execution of multi-task learning algorithms on resource-constrained devices by integrating advanced software frameworks alongside model quantization. This quantization effectively balanced accuracy, performance, and memory demands. The approach optimized energy efficiency and maintained performance integrity through a well-coordinated hardware and software design, enabling the simultaneous execution of complex tasks. Moreover, innovations in memory allocation within the custom DPU-TRD architecture led to a performance of 24,715 GOP with 4% lower power consumption and 18.86% less logic resource consumption compared to similar studies. This demonstrates the study's significant advancements in optimizing resource use and improving computational efficiency in A/DAS applications.

7 Future Work

Continuing the advancement of FPGA-based MPSoC systems and low-precision QNNs presents exciting opportunities for enhancing computational performance and efficiency across various applications. Future research should focus on advancing quantization techniques and exploring more sophisticated quantization levels to refine the balance between computational efficiency and model accuracy. This involves investigating the impact of different quantization depths on performance in diverse scenarios such as autonomous

driving and real-time video processing.

There is also significant potential in extending the application of these technologies to other data-intensive domains like competent healthcare, IoT, and robotics. Adapting and optimizing the hardware and software architectures for specific tasks within these fields could demonstrate the versatility and scalability of FPGA-based and quantized models.

Moreover, further improvements in energy efficiency are crucial. Developing advanced power management techniques, including adaptive power scaling and energy-aware task scheduling, could enhance these systems' sustainability and operational efficiency. Additionally, enhancing the integration of AI algorithms directly into FPGA and MPSoC designs could lead to processing speed and efficiency breakthroughs, particularly by developing dedicated hardware accelerators for specific AI functions. Finally, comprehensive system validation is essential to ensure the reliability and robustness of these systems across different platforms and operational conditions. Collaborative efforts in developing open-source frameworks that facilitate these technologies' design, testing, and deployment could accelerate innovation and standardization in the field.

References

[1] Suparshya Babu Sukhavasi, Susrutha Babu Sukhavasi, Khaled Elleithy, Ahmed El-Sayed, and Abdelrahman Elleithy. A hybrid model for driver emotion detection using feature fusion approach. International journal of environmental research and public health, 19(5):3085, 2022.

[2] Ashish Patel. Machine learning algorithm overview, Jul 2018.

[3] Wanjun Xia, Yanping Jiang, Xiaohong Chen, and Rui Zhao. Application of machine learning algorithms in municipal solid waste management: A mini review. Waste Management & Research, 40(6):609–624, 2022.

[4] Yanzhou Fu, Austin RJ Downey, Lang Yuan, Tianyu Zhang, Avery Pratt, and Yunusa Balogun. Machine learning algorithms for defect detection in metal laser-based additive manufacturing: a review. Journal of Manufacturing Processes, 75:693–710, 2022.

[5] Jeffrey A. Fessler. Fessler, Jeffrey A.: UM EECS Web page. — web.eecs.umich.edu. https://web.eecs.umich.edu/~fessler/. [Accessed 29-03-2024].

[6] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. pages 21–37, 2016.

[7] TK Balaji, Chandra Sekhara Rao Annavarapu, and Annushree Bablani. Machine learning algorithms for social media analysis: A survey. Computer Science Review, 40:100395, 2021.

[8] Jonas Fuchs, Markus Gardill, Maximilian Lübke, Anand Dubey, and Fabian Lurz. A machine learning perspective on automotive radar direction of arrival estimation. IEEE Access, 10:6775–6797, 2022.

[9] Mozhgan Nasr Azadani and Azzedine Boukerche. Driverrep: Driver identification through driving behavior embeddings. Journal of Parallel and Distributed Computing, 162:105–117, 2022.

[10] Christine Dewi, Rung-Ching Chen, Yan-Ting Liu, and Shao-Kuo Tai. Synthetic data generation using dcgan for improved traffic sign recognition. Neural Computing and Applications, 34(24):21465–21480, 2022.

[11] Shahid Latif, Maha Driss, Wadii Boulila, Sajjad Shaukat Jamal, Zeba Idrees, Jawad Ahmad, et al. Deep learning for the industrial internet of things (iiot): A comprehensive survey of techniques, implementation frameworks, potential applications, and future directions. Sensors, 21(22):7518, 2021.

[12] Radwa Elshawi, Abdul Wahab, Ahmed Barnawi, and Sherif Sakr. Dlbench: a comprehensive experimental evaluation of deep learning frameworks. Cluster Computing, 24(3):2017–2038, 2021.

[13] Wang Xiangxue, Xu Lunhui, and Chen Kaixun. Data-driven short-term forecasting for urban road network traffic based on data processing and lstm-rnn. Arabian Journal for Science and Engineering, 44(4):3043–3060, 2019.

[14] Shidrokh Goudarzi, Mohd Nazri Kama, Mohammad Hossein Anisi, Seyed Ahmad Soleymani, and Faiyaz Doctor. Self-organizing traffic flow prediction with an optimized deep belief network for internet of vehicles. Sensors, 18(10):3459, 2018.

[15] Juan Luis Pérez, Alberto Gutierrez-Torre, Josep Ll Berral, and David Carrera. A resilient and distributed near real-time traffic forecasting application for fog computing environments. Future Generation Computer Systems, 87:198–212, 2018.

[16] Zhongjian Lv, Jiajie Xu, Kai Zheng, Hongzhi Yin, Pengpeng Zhao, and Xiaofang Zhou. Lc-rnn: A deep learning model for traffic speed prediction. In IJCAI, volume 2018, page 27th, 2018.

[17] Dinesh Singh and Chalavadi Krishna Mohan. Deep spatio-temporal representation for detection of road accidents using stacked autoencoder. IEEE Transactions on Intelligent Transportation Systems, 20(3):879–887, 2018.

[18] Dr. Steve Arar. FIR Filter Design by Windowing: Concepts and the Rectangular Window. https://www.allaboutcircuits.com/technical-articles/design-examples-of-fir-filters-using-window-method/. [Accessed 29-03-2024].

[19] Dave Evans. The internet of things: How the next evolution of the internet is changing everything. CISCO white paper, 1(2011):1–11, April 2011.

[20] Cisco. Annual internet report (2018-2023). White Paper. Available online: https://www.cisco.com/c/en/us/solutions/collateral/executive-perspectives/annual-internet-report/white-paper-c11-741490.html (accessed on February 2021).

[21] Sérgio Branco, André G Ferreira, and Jorge Cabral. Machine learning in resource-scarce embedded systems, fpgas, and end-devices: A survey. Electronics, 8(11):1289, 2019.

[22] Gareth Mitchell. How much data is on the internet? Science Focus (The Home of BBC Science Focus Magazine).[Online]. https://www. sciencefocus. com/future-technology/how-much-data-is-on-the-internet/. Accessed, 20, 2019.

[23] C. Thomas M. Patel, J. Shangkuan. What's new with the internet of things?

[24] Mehryar Mohri, Afshin Rostamizadeh, and Ameet Talwalkar. Foundations of machine learning.[sl], 2012.

[25] Sumit Das, Aritra Dey, Akash Pal, and Nabamita Roy. Applications of artificial intelligence in machine learning: review and prospect. International Journal of Computer Applications, 115(9), 2015.

[26] J. Brownlee. Machine learning mastery. Available online: https://machinelearningmastery.com/types-of-classification-in-machine-learning, April 2020. accessed on February 2021.

[27] Abdallah Moubayed, Mohammadnoor Injadat, Ali Bou Nassif, Hanan Lutfiyya, and Abdallah Shami. E-learning: Challenges and research opportunities using machine learning & data analytics. IEEE Access, 6:39117–39138, 2018.

[28] A. Prabhat and V. Khullar. Sentiment classification on big data using naïve bayes and logistic regression. In 2017 International Conference on Computer Communication and Informatics (ICCCI), pages 1–5, 2017.

[29] Gopi Battineni, Nalini Chintalapudi, and Francesco Amenta. Machine learning in medicine: Performance calculation of dementia prediction by support vector machines (svm). Informatics in Medicine Unlocked, 16:100200, 2019.

[30] Martin T Hagan, Howard B Demuth, and Mark Beale. Neural network design. PWS Publishing Co., 1997.

[31] Mingzhe Chen, Ursula Challita, Walid Saad, Changchuan Yin, and Mérouane Debbah. Artificial neural networks-based machine learning for wireless networks: A tutorial. IEEE Communications Surveys & Tutorials, 21(4):3039–3071, 2019.

[32] Qiming Zhang, Haoyi Yu, Martina Barbiero, Baokai Wang, and Min Gu. Artificial neural networks enabled by nanophotonics. Light: Science & Applications, 8(1):1–14, 2019.

[33] Maryam Tayefi, Mohammad Tajfard, Sara Saffar, Parichehr Hanachi, Ali Reza Amirabadizadeh, Habibollah Esmaeily, Ali Taghipour, Gordon A Ferns, Mohsen Moohebati, and Majid Ghayour-Mobarhan. hs-crp is strongly associated with coronary heart disease (chd): A data mining approach using decision tree algorithm. Computer methods and programs in biomedicine, 141:105–109, 2017.

[34] Oracle Help Center. Data mining concepts, regression.

[35] Amand F Schmidt and Chris Finan. Linear regression and the normality assumption. Journal of clinical epidemiology, 98:146–151, 2018.

[36] Athanasios Voulodimos, Nikolaos Doulamis, Anastasios Doulamis, and Eftychios Protopapadakis. Deep learning for computer vision: A brief review. Computational intelligence and neuroscience, 2018, 2018.

[37] Andreas Kamilaris and Francesc X Prenafeta-Boldú. Deep learning in agriculture: A survey. Computers and electronics in agriculture, 147:70–90, 2018.

[38] Erick Moen, Dylan Bannon, Takamasa Kudo, William Graf, Markus Covert, and David Van Valen. Deep learning for cellular image analysis. Nature methods, 16(12):1233–1246, 2019.

[39] Warren S McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4):115–133, 1943.

[40] Zachary C Lipton, John Berkowitz, and Charles Elkan. A critical review of recurrent neural networks for sequence learning. arXiv preprint arXiv:1506.00019, 2015.

[41] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. Neural computation, 18(7):1527–1554, 2006.

[42] Wanli Ouyang, Xingyu Zeng, Xiaogang Wang, Shi Qiu, Ping Luo, Yonglong Tian, Hongsheng Li, Shuo Yang, Zhe Wang, Hongyang Li, et al. Deepid-net: Object detection with deformable part based convolutional neural networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(7):1320–1334, 2016.

[43] Ali Diba, Vivek Sharma, Ali Pazandeh, Hamed Pirsiavash, and Luc Van Gool. Weakly supervised cascaded convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 914–922, 2017.

[44] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. In Proceedings of the IEEE international conference on computer vision, pages 1520–1528, 2015.

[45] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3431–3440, 2015.

[46] Nikolaos Doulamis and Athanasios Voulodimos. Fast-mdl: Fast adaptive supervised training of multi-layered deep learning models for consistent object tracking and classification. In 2016 IEEE International Conference on Imaging Systems and Techniques (IST), pages 318–323. IEEE, 2016.

[47] Nikolaos Doulamis. Adaptable deep learning structures for object labeling/tracking under dynamic visual environments. Multimedia Tools and Applications, 77(8):9651–9689, 2018.

[48] Rikiya Yamashita, Mizuho Nishio, Richard Kinh Gian Do, and Kaori Togashi. Convolutional neural networks: an overview and application in radiology. Insights into imaging, 9(4):611–629, 2018.

[49] Long Wen, Xinyu Li, Liang Gao, and Yuyan Zhang. A new convolutional neural network-based data-driven fault diagnosis method. IEEE Transactions on Industrial Electronics, 65(7):5990–5998, 2017.

[50] Tianmei Guo, Jiwen Dong, Henjian Li, and Yunxing Gao. Simple convolutional neural network on image classification. In 2017 IEEE 2nd International Conference on Big Data Analysis (ICBDA), pages 721–724. IEEE, 2017.

[51] Saad Albawi, Tareq Abed Mohammed, and Saad Al-Zawi. Understanding of a convolutional neural network. In 2017 International Conference on Engineering and Technology (ICET), pages 1–6. Ieee, 2017.

[52] U Rajendra Acharya, Shu Lih Oh, Yuki Hagiwara, Jen Hong Tan, Muhammad Adam, Arkadiusz Gertych, and Ru San Tan. A deep convolutional neural network model to classify heartbeats. Computers in biology and medicine, 89:389–396, 2017.

[53] Peng Yao, Huaqiang Wu, Bin Gao, Jianshi Tang, Qingtian Zhang, Wenqiang Zhang, J Joshua Yang, and He Qian. Fully hardware-implemented memristor convolutional neural network. Nature, 577(7792):641–646, 2020.

[54] Larry Medsker and Lakhmi C Jain. Recurrent neural networks: design and applications. CRC press, 1999.

[55] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In ICML, 2011.

[56] Lichao Mou, Pedram Ghamisi, and Xiao Xiang Zhu. Deep recurrent neural networks for hyperspectral image classification. IEEE Transactions on Geoscience and Remote Sensing, 55(7):3639–3655, 2017.

[57] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. A review of recurrent neural networks: Lstm cells and network architectures. Neural computation, 31(7):1235–1270, 2019.

[58] A. Graves, A. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In 2013 IEEE International Conference on Acoustics, Speech and Signal Processing, pages 6645–6649, Vancouver, BC, Canada, 2013.

[59] Hojjat Salehinejad, Sharan Sankar, Joseph Barfett, Errol Colak, and Shahrokh Valaee. Recent advances in recurrent neural networks. arXiv preprint arXiv:1801.01078, 2017.

[60] Ersin Aytaç. Unsupervised learning approach in defining the similarity of catchments: Hydrological response unit based k-means clustering, a demonstration on western black sea region of turkey. International Soil and Water Conservation Research, 8(3):321–331, 2020.

[61] Kristina P Sinaga and Miin-Shen Yang. Unsupervised k-means clustering algorithm. IEEE Access, 8:80716–80727, 2020.

[62] Jörg Lücke and Dennis Forster. k-means as a variational em approximation of gaussian mixture models. Pattern Recognition Letters, 125:349–356, 2019.

[63] Ce Wang, Zhangling Chen, Kun Shang, and Huaming Wu. Label-removed generative adversarial networks incorporating with k-means. Neurocomputing, 361:126–136, 2019.

[64] Paulene Govender and Venkataraman Sivakumar. Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980–2019). Atmospheric Pollution Research, 11(1):40–56, 2020.

[65] Xinzheng Xu, Tianming Liang, Jiong Zhu, Dong Zheng, and Tongfeng Sun. Review of classical dimensionality reduction and sample selection methods for large-scale data processing. Neurocomputing, 328:5–15, 2019.

[66] Laurens Van Der Maaten, Eric Postma, Jaap Van den Herik, et al. Dimensionality reduction: a comparative. J Mach Learn Res, 10(66-71):13, 2009.

[67] Jason Kane Gilbertson and Adriaan Van Niekerk. Value of dimensionality reduction for crop differentiation with multi-temporal imagery and machine learning. Computers and Electronics in Agriculture, 142:50–58, 2017.

[68] G Thippa Reddy, M Praveen Kumar Reddy, Kuruva Lakshmanna, Rajesh Kaluri, Dharmendra Singh Rajput, Gautam Srivastava, and Thar Baker. Analysis of dimensionality reduction techniques on big data. IEEE Access, 8:54776–54788, 2020.

[69] Hervé Abdi and Lynne J Williams. Principal component analysis. Wiley interdisciplinary reviews: computational statistics, 2(4):433–459, 2010.

[70] Aloke Datta, Susmita Ghosh, and Ashish Ghosh. Pca, kernel pca and dimensionality reduction in hyperspectral images. In Advances in Principal Component Analysis, pages 19–46. Springer, 2018.

[71] Jesper E Van Engelen and Holger H Hoos. A survey on semi-supervised learning. Machine Learning, 109(2):373–440, 2020.

[72] Madhav Sigdel, İmren Dinç, Semih Dinç, Madhu S Sigdel, Marc L Pusey, and Ramazan S Aygün. Evaluation of semi-supervised learning for classification of protein crystallization imagery. In IEEE SOUTHEASTCON 2014, pages 1–6. IEEE, 2014.

[73] Ioannis E Livieris, Andreas Kanavos, Vassilis Tampakas, and Panagiotis Pintelas. An auto-adjustable semi-supervised self-training algorithm. Algorithms, 11(9):139, 2018.

[74] Jafar Tanha, Maarten van Someren, and Hamideh Afsarmanesh. Semi-supervised self-training for decision tree classifiers. International Journal of Machine Learning and Cybernetics, 8(1):355–370, 2017.

[75] Xiaojin Zhu and Andrew B Goldberg. Introduction to semi-supervised learning. Synthesis lectures on artificial intelligence and machine learning, 3(1):1–130, 2009.

[76] Michael C Burkhart and Kyle Shan. Deep low-density separation for semi-supervised classification. In International Conference on Computational Science, pages 297–311. Springer, 2020.

[77] Thomas M Moerland, Joost Broekens, and Catholijn M Jonker. Model-based reinforcement learning: A survey. arXiv preprint arXiv:2006.16712, 2020.

[78] Kai Arulkumaran, Marc Peter Deisenroth, Miles Brundage, and Anil Anthony Bharath. A brief survey of deep reinforcement learning. arXiv preprint arXiv:1708.05866, 2017.

[79] Nguyen Cong Luong, Dinh Thai Hoang, Shimin Gong, Dusit Niyato, Ping Wang, Ying-Chang Liang, and Dong In Kim. Applications of deep reinforcement learning in communications and networking: A survey. IEEE Communications Surveys & Tutorials, 21(4):3133–3174, 2019.

[80] Christian Wirth, Riad Akrour, Gerhard Neumann, Johannes Fürnkranz, et al. A survey of preference-based reinforcement learning methods. Journal of Machine Learning Research, 18(136):1–46, 2017.

[81] Jelena Luketina, Nantas Nardelli, Gregory Farquhar, Jakob Foerster, Jacob Andreas, Edward Grefenstette, Shimon Whiteson, and Tim Rocktäschel. A survey of reinforcement learning informed by natural language. arXiv preprint arXiv:1906.03926, 2019.

[82] Lucian Busoniu, Robert Babuska, Bart De Schutter, and Damien Ernst. Reinforcement learning and dynamic programming using function approximators. CRC press, 2017.

[83] A. Sharma. Role of dynamic programming in machine learning aim.

[84] Yuki Nagai, Huitao Shen, Yang Qi, Junwei Liu, and Liang Fu. Self-learning monte carlo method: Continuous-time algorithm. Physical Review B, 96(16):161102, 2017.

[85] Analytics Vidhya A. CHOUDHARY. Reinforcement learning: Introduction to monte carlo learning using the openai gym toolkit".

[86] Juan Borrego-Carazo, David Castells-Rufas, Ernesto Biempica, and Jordi Carrabina. Resource-constrained machine learning for adas: a systematic review. IEEE Access, 8:40573–40598, 2020.

[87] Linda Capito and Keith A. Redmill. Methodology for hazard identification and mitigation strategies applied to an overtaking assistant adas. In 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), pages 3972–3977, 2021.

[88] Tyler GR Reid, Andrew Neish, and Brian Manning. Localization & mapping requirements for level 2+ autonomous vehicles. arXiv preprint arXiv:2212.08231, 2022.

[89] Rami Debouk et al. Overview of the 2nd edition of iso 26262: Functional safety-road vehicles. General Motors Company, Warren, MI, USA, 2018.

[90] Rob Palin, David Ward, Ibrahim Habli, and Roger Rivett. Iso 26262 safety cases: Compliance and assurance. 2011.

[91] International Organization for Standardization. ISO 26262-9:2018 — iso.org. https://www.iso.org/standard/68391.html, 2018. [Accessed 29-Dec-2022].

[92] Khan Muhammad, Amin Ullah, Jaime Lloret, Javier Del Ser, and Victor Hugo C. de Albuquerque. Deep learning for safe autonomous driving: Current challenges and future directions. IEEE Transactions on Intelligent Transportation Systems, 22(7):4316–4336, 2021.

[93] B Ravi Kiran, Ibrahim Sobh, Victor Talpaert, Patrick Mannion, Ahmad A. Al Sallab, Senthil Yogamani, and Patrick Pérez. Deep reinforcement learning for autonomous driving: A survey. IEEE Transactions on Intelligent Transportation Systems, 23(6):4909–4926, 2022.

[94] Éloi Zablocki, Hédi Ben-Younes, Patrick Pérez, and Matthieu Cord. Explainability of deep vision-based autonomous driving systems: Review and challenges. International Journal of Computer Vision, 130(10):2425–2452, 2022.

[95] Monirul Islam Pavel, Siok Yee Tan, and Azizi Abdullah. Vision-based autonomous vehicle systems based on deep learning: A systematic literature review. Applied Sciences, 12(14), 2022.

[96] Jules Karangwa, Jun Liu, and Zixuan Zeng. Vehicle detection for autonomous driving: A review of algorithms and datasets. IEEE Transactions on Intelligent Transportation Systems, 24(11):11568–11594, 2023.

[97] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. 2015.

[98] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. pages 1–9, 2015.

[99] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778, 2016.

[100] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In Proceedings of the IEEE international conference on computer vision, pages 2961–2969, 2017.

[101] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017.

[102] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You only look once: Unified, real-time object detection. pages 779–788, 2016.

[103] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. Learning deconvolution network for semantic segmentation. pages 1520–1528, 2015.

[104] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(12):2481–2495, 2017.

[105] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Towards end-to-end lane detection: an instance segmentation approach. In 2018 IEEE intelligent vehicles symposium (IV), pages 286–291. IEEE, 2018.

[106] Seokju Lee, Junsik Kim, Jae Shin Yoon, Seunghak Shin, Oleksandr Bailo, Namil Kim, Tae-Hee Lee, Hyun Seok Hong, Seung-Hoon Han, and In So Kweon. Vpgnet: Vanishing point guided network for lane and road marking detection and recognition. pages 1947–1955, 2017.

[107] Yann LeCun, Lawrence D Jackel, Léon Bottou, Corinna Cortes, John S Denker, Harris Drucker, Isabelle Guyon, Urs A Muller, Eduard Sackinger, Patrice Simard, et al. Learning algorithms for classification: A comparison on handwritten digit recognition. Neural networks: the statistical mechanics perspective, 261(276):2, 1995.

[108] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. Neural computation, 1(4):541–551, 1989.

[109] Gaurav Nakhare. Hardware options for machine/deep learning.

[110] P. Jawandhiya. Hardware design for machine learning. International Journal of Artificial Intelligence & Applications, 9:63–84, 2018.

[111] Davy Neven, Bert De Brabandere, Stamatios Georgoulis, Marc Proesmans, and Luc Van Gool. Fast scene understanding for autonomous driving. arXiv preprint arXiv:1708.02550, 2017.

[112] Marvin Teichmann, Michael Weber, Marius Zoellner, Roberto Cipolla, and Raquel Urtasun. Multinet: Real-time joint semantic reasoning for autonomous driving. In 2018 IEEE intelligent vehicles symposium (IV), pages 1013–1020. IEEE, 2018.

[113] Jinzhang Peng, Lu Tian, Xijie Jia, Haotian Guo, Yongsheng Xu, Dongliang Xie, Hong Luo, Yi Shan, and Yu Wang. Multi-task adas system on fpga. In 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), pages 171–174. IEEE, 2019.

[114] Abdallah Moujahid, Mounir ElAraki Tantaoui, Manolo Dulva Hina, Assia Soukane, Andrea Ortalda, Ahmed ElKhadimi, and Amar Ramdane-Cherif. Machine learning techniques in adas: a review. In 2018 International Conference on Advances in Computing and Communication Engineering (ICACCE), pages 235–242. IEEE, 2018.

[115] Fadi Al Machot, Mouhannad Ali, Ahmad Haj Mosa, Christopher Schwarzlmüller, Markus Gutmann, and Kyandoghere Kyamakya. Real-time raindrop detection based on cellular neural networks for adas. Journal of Real-Time Image Processing, 16(4):931–943, 2019.

[116] Mahshid Helali Moghadam, Markus Borg, Mehrdad Saadatmand, Seyed Jalaleddin Mousavirad, Markus Bohlin, and Björn Lisper. Machine learning testing in an adas case study using simulation-integrated bio-inspired search-based testing. arXiv preprint arXiv:2203.12026, 2022.

[117] Jiaxiang Shi, Xiaohua Tian, Zhiyuan Zheng, and Tao Zhang. Application research of cnn accelerator design based on fpga in adas. In IOP Conference Series: Materials Science and Engineering, volume 768, page 072014. IOP Publishing, 2020.

[118] Iqbal H Sarker. Machine learning: Algorithms, real-world applications and research directions. SN Computer Science, 2(3):1–21, 2021.

[119] Jaswanth Nidamanuri, Chinmayi Nibhanupudi, Rolf Assfalg, and Hrishikesh Venkataraman. A progressive review-emerging technologies for adas driven solutions. IEEE Transactions on Intelligent Vehicles, 2021.

[120] Agnès Ghorbel, Nader Ben Amor, and Mohamed Abid. Gpgpu-based parallel computing of viola and jones eyes detection algorithm to drive an intelligent wheelchair. Journal of Signal Processing Systems, 94(12):1365–1379, 2022.

[121] MR Rani, MZC Mustafar, NHF Ismail, MSF Mansor, and Z Zainuddin. Road peculiarities detection using deep learning for vehicle vision system. In IOP Conference Series: Materials Science and Engineering, volume 1068, page 012001. IOP Publishing, 2021.

[122] Alvaro Hernández, Suhan Woo, Hector Corrales, Ignacio Parra, Euntai Kim, David Fernández Llorca, and MA Sotelo. 3d-deep: 3-dimensional deep-learning based on elevation patterns for road scene interpretation. In 2020 IEEE Intelligent Vehicles Symposium (IV), pages 892–898. IEEE, 2020.

[123] Mihail-Alexandru Andrei, Costin-Anton Boiangiu, Nicolae Tarbă, and Mihai-Lucian Voncilă. Robust lane detection and tracking algorithm for steering assist systems. Machines, 10(1):10, 2021.

[124] Yiman Chen, Zhiyu Xiang, and Wentao Du. Improving lane detection with adaptive homography prediction. The Visual Computer, 39(2):581–595, 2023.

[125] Tiago Almeida, Bernardo Lourenço, and Vitor Santos. Road detection based on simultaneous deep learning approaches. Robotics and Autonomous Systems, 133:103605, 2020.

[126] Jaehoon Cho, Youngjung Kim, Hyungjoo Jung, Changjae Oh, Jaesung Youn, and Kwanghoon Sohn. Multi-task self-supervised visual representation learning for monocular road segmentation. In 2018 ieee international conference on multimedia and expo (icme), pages 1–6. IEEE, 2018.

[127] Michal Machura, Michal Danilowicz, and Tomasz Kryjak. Embedded object detection with custom littlenet, finn and vitis ai dcnn accelerators. Journal of Low Power Electronics and Applications, 12(2):30, 2022.

[128] Neerav Sharma and Rahul Dev Garg. Cost reduction for advanced driver assistance systems through hardware downscaling and deep learning. Systems Engineering, 25(2):133–143, 2022.

[129] Emin Güney, Cüneyt Bayilmiş, and Batuhan Çakan. An implementation of real-time traffic signs and road objects detection based on mobile gpu platforms. IEEE Access, 10:86191–86203, 2022.

[130] Muhammad Ali Farooq, Peter Corcoran, Cosmin Rotariu, and Waseem Shariff. Object detection in thermal spectrum for advanced driver-assistance systems (adas). IEEE Access, 9:156465–156481, 2021.

[131] Hsiang-Yu Han, Yu-Chi Chen, Pei-Yung Hsiao, and Li-Chen Fu. Using channel-wise attention for deep cnn based real-time semantic segmentation with class-aware edge information. IEEE Transactions on Intelligent Transportation Systems, 22(2):1041–1051, 2020.

[132] Srivatsan Krishnan, Zishen Wan, Kshitij Bhardwaj, Paul Whatmough, Aleksandra Faust, Sabrina Neuman, Gu-Yeon Wei, David Brooks, and Vijay Janapa Reddi. Automatic domain-specific soc design for autonomous unmanned aerial vehicles. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO), pages 300–317. IEEE, 2022.

[133] Pedro Azevedo and Vítor Santos. Comparative analysis of multiple yolo-based target detectors and trackers for adas in edge devices. Robotics and Autonomous Systems, 171:104558, 2024.

[134] Hao Liu, Dong Li, JinZhang Peng, Qingjie Zhao, Lu Tian, and Yi Shan. Mtnas: search multi-task networks for autonomous driving. In Proceedings of the Asian Conference on Computer Vision, 2020.

[135] Flora Dellinger, Thomas Boulay, Diego Mendoza Barrenechea, Said El-Hachimi, Isabelle Leang, and Fabian Bürger. Multi-task network pruning and embedded optimization for real-time deployment in adas. arXiv preprint arXiv:2101.07831, 2021.

[136] Stavros Kalapothas, Georgios Flamis, and Paris Kitsos. Efficient edge-ai application deployment for fpgas. Information, 13(6):279, 2022.

[137] Shaoxia Fang, Lu Tian, Junbin Wang, Shuang Liang, Dongliang Xie, Zhongmin Chen, Lingzhi Sui, Qian Yu, Xiaoming Sun, Yi Shan, et al. Real-time object detection and semantic segmentation hardware system with deep learning networks. In 2018 International conference on field-programmable technology (FPT), pages 389–392. IEEE, 2018.

[138] Chun-Yu Lai, Bo-Xun Wu, Vinay Malligere Shivanna, and Jiun-In Guo. Mtsan: Multi-task semantic attention network for adas applications. IEEE Access, 9:50700–50714, 2021.

[139] Guner Tatar and Salih Bayar. Real-time multi-task adas implementation on reconfigurable heterogeneous mpsoc architecture. IEEE Access, 11:80741–80760, 2023.

[140] Masayuki Miyama. Robust inference of multi-task convolutional neural network for advanced driving assistance by embedding coordinates. World Congress on Electrical Engineering and Computer Systems and Science, 2022.

[141] S. Evanczuk. Applying machine learning in embedded systems.

[142] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. arXiv preprint arXiv:1409.1556, 2014.

[143] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 770–778, 2016.

[144] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1–9, 2015.

[145] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 2818–2826, 2016.

[146] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[147] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. Advances in neural information processing systems, 25:1097–1105, 2012.

[148] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4700–4708, 2017.

[149] Claudine Badue, Rânik Guidolini, Raphael Vivacqua Carneiro, Pedro Azevedo, Vinicius B Cardoso, Avelino Forechi, Luan Jesus, Rodrigo Berriel, Thiago M Paixao, Filipe Mutz, et al. Self-driving cars: A survey. Expert Systems with Applications, 165:113816, 2021.

[150] Eve Mitleton-Kelly, Ivan Deschenaux, Christian Maag, Matthew Fullerton, and Nihan Celikkaya. Enhancing crowd evacuation and traffic management through ami technologies: A review of the literature. Co-evolution of intelligent socio-technical systems, pages 19–41, 2013.

[151] Shane Tuohy, Martin Glavin, Ciarán Hughes, Edward Jones, Mohan Trivedi, and Liam Kilmartin. Intra-vehicle networks: A review. IEEE Transactions on Intelligent Transportation Systems, 16(2):534–545, 2014.

[152] Adam Ziebinski, Rafal Cupek, Damian Grzechca, and Lukas Chruszczyk. Review of advanced driver assistance systems (adas). In AIP Conference Proceedings, volume 1906, page 120002. AIP Publishing LLC, 2017.

[153] Adam Ziebinski, Rafal Cupek, Hueseyin Erdogan, and Sonja Waechter. A survey of adas technologies for the future perspective of sensor fusion. In International Conference on Computational Collective Intelligence, pages 135–146. Springer, 2016.

[154] Harsha Jakkanahalli Vishnukumar, Björn Butting, Christian Müller, and Eric Sax. Machine learning and deep neural network—artificial intelligence core for lab and real-world test and validation for adas and autonomous vehicles: Ai for efficient and quality test and validation. In 2017 Intelligent Systems Conference (IntelliSys), pages 714–721. IEEE, 2017.

[155] Klaus Bengler, Klaus Dietmayer, Berthold Farber, Markus Maurer, Christoph Stiller, and Hermann Winner. Three decades of driver assistance systems: Review and future perspectives. IEEE Intelligent transportation systems magazine, 6(4):6–22, 2014.

[156] Agus Mulyanto, Rohmat Indra Borman, Purwono Prasetyawan, Wisnu Jatmiko, Petrus Mursanto, and Aprian Sinaga. Indonesian traffic sign recognition for advanced driver assistent (adas) using yolov4. In 2020 3rd International Seminar on Research of Information Technology and Intelligent Systems (ISRITI), pages 520–524. IEEE, 2020.

[157] Hee Seok Lee and Kang Kim. Simultaneous traffic sign detection and boundary estimation using convolutional neural network. IEEE Transactions on Intelligent Transportation Systems, 19(5):1652–1663, 2018.

[158] Po-Cheng Shih, Chi-Yi Tsai, and Chun-Fei Hsu. An efficient automatic traffic sign detection and recognition method for smartphones. In 2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI), pages 1–5, 2017.

[159] Helder Novais and António Ramires Fernandes. Community based repository for georeferenced traffic signs. In 2017 24º Encontro Português de Computação Gráfica e Interação (EPCGI), pages 1–8. IEEE, 2017.

[160] Wei Liu, Shuang Li, Jin Lv, Bing Yu, Ting Zhou, Huai Yuan, and Hong Zhao. Real-time traffic light recognition based on smartphone platforms. IEEE Transactions on Circuits and Systems for Video Technology, 27(5):1118–1131, 2016.

[161] Nizar Akbar Arifin, Budhi Irawan, and Casi Setianingsih. Traffic sign recognition application using speeded-up robust features (surf) and support vector machine (svm) based on android. In 2017 IEEE Asia Pacific Conference on Wireless and Mobile (APWiMob), pages 52–58. IEEE, 2017.

[162] Hatem Ibrahem, Ahmed Salem, and Hyun Soo Kang. Weakly supervised traffic sign detection in real time using single cnn architecture for multiple purposes. In 2020 IEEE International Conference on Consumer Electronics (ICCE), pages 1–4, 2020.

[163] Truong Quang Vinh. Real-time traffic sign detection and recognition system based on friendlyarm tiny4412 board. In 2015 International Conference on Communications, Management and Telecommunications (ComManTel), pages 142–146. IEEE, 2015.

[164] Jiayuan Yu, Huiling Liu, and Huayan Zhang. Research on detection and recognition algorithm of road traffic signs. In 2019 Chinese Control And Decision Conference (CCDC), pages 1996–2001. IEEE, 2019.

[165] Shyam Jagannathan, Kumar Desappan, Pramod Swami, Manu Mathew, Soyeb Nagori, Kedar Chitnis, Yogesh Marathe, Deepak Poddar, Suriya Narayanan, and Anshu Jain. Efficient object detection and classification on low power embedded systems. In 2017 IEEE International Conference on Consumer Electronics (ICCE), pages 233–234. IEEE, 2017.

[166] Yuteng Zhou, Zhilu Chen, and Xinming Huang. A system-on-chip fpga design for real-time traffic signal recognition system. In 2016 IEEE International Symposium on Circuits and Systems (ISCAS), pages 1778–1781. IEEE, 2016.

[167] BB Shabarinath and P Muralidhar. Convolutional neural network based traffic-sign classifier optimized for edge inference. In 2020 IEEE REGION 10 CONFERENCE (TENCON), pages 420–425. IEEE, 2020.

[168] Wajdi Farhat, Souhir Sghaier, Hassene Faiedh, and Chokri Souani. Design of efficient embedded system for road sign recognition. Journal of Ambient Intelligence and Humanized Computing, 10(2):491–507, 2019.

[169] Wenzheng Hu, Qing Zhuo, Changshui Zhang, and Jianke Li. Fast branch convolutional neural network for traffic sign recognition. IEEE Intelligent Transportation Systems Magazine, 9(3):114–126, 2017.

[170] Yan Han, Kushal Virupakshappa, Esdras Vitor Silva Pinto, and Erdal Oruklu. Hardware/software co-design of a traffic sign recognition system using zynq fpgas. Electronics, 4(4):1062–1089, 2015.

[171] Wajdi Farhat, Hassene Faiedh, Chokri Souani, and Kamel Besbes. Real-time embedded system for traffic sign recognition based on zedboard. Journal of Real-Time Image Processing, 16(5):1813–1823, 2019.

[172] Huai-Mao Weng and Ching-Te Chiu. Resource efficient hardware implementation for real-time traffic sign recognition. In 2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pages 1120–1124, 2018.

[173] Yan Han, Kushal Virupakshappa, and Erdal Oruklu. Robust traffic sign recognition with feature extraction and k-nn classification methods. In 2015 IEEE International Conference on Electro/Information Technology (EIT), pages 484–488. IEEE, 2015.

[174] Chi-Yi Tsai, Hsien-Chen Liao, and Kuang-Jui Hsu. Real-time embedded implementation of robust speed-limit sign recognition using a novel centroid-to-contour description method. IET Computer Vision, 11(6):407–414, 2017.

[175] Samuel L Gomes, Elizângela de S Rebouças, Edson Cavalcanti Neto, João P Papa, Victor HC de Albuquerque, Pedro P Rebouças Filho, and Joao Manuel RS Tavares. Embedded real-time speed limit sign recognition using image processing and machine learning techniques. Neural Computing and Applications, 28(1):573–584, 2017.

[176] Anass BARODI, Abderrahim Bajit, Abdelkarim ZEMMOURI, Mohammed Benbrahim, and Ahmed Tamtaoui. Improved deep learning performance for real-time traffic sign detection and recognition applicable to intelligent transportation systems. International Journal of Advanced Computer Science and Applications, 13(5), 2022.

[177] Christine Dewi, Rung-Ching Chen, Xiaoyi Jiang, and Hui Yu. Deep convolutional neural network for enhancing traffic sign recognition developed on yolo v4. Multimedia Tools and Applications, pages 1–25, 2022.

[178] Duy Tran, Ha Manh Do, Weihua Sheng, He Bai, and Girish Chowdhary. Real-time detection of distracted driving based on deep learning. IET Intelligent Transport Systems, 12(10):1210–1219, 2018.

[179] Guido Borghi, Riccardo Gasparini, Roberto Vezzani, and Rita Cucchiara. Embedded recurrent network for head pose estimation in car. In 2017 IEEE Intelligent Vehicles Symposium (IV), pages 1503–1508, 2017.

[180] Wenjie Song, Yi Yang, Mengyin Fu, Yujun Li, and Meiling Wang. Lane detection and classification for forward collision warning system based on stereo vision. IEEE Sensors Journal, 18(12):5151–5163, 2018.

[181] Qin Zou, Hanwen Jiang, Qiyu Dai, Yuanhao Yue, Long Chen, and Qian Wang. Robust lane detection from continuous driving scenes using deep neural networks. IEEE transactions on vehicular technology, 69(1):41–54, 2019.

[182] Yuteng Zhou, Yecheng Lyu, and Xinming Huang. Roadnet: An 80-mw hardware accelerator for road detection. IEEE Embedded Systems Letters, 11(1):21–24, 2018.

[183] Jung-Gu Kim and Jae-Hyung Yoo. Hw implementation of real-time road & lane detection in fpga-based stereo camera. In 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), pages 1–4. IEEE, 2019.

[184] Yecheng Lyu, Lin Bai, and Xinming Huang. Real-time road segmentation using lidar data processing on an fpga. In 2018 IEEE International Symposium on Circuits and Systems (ISCAS), pages 1–5. IEEE, 2018.

[185] Zhenpeng Chen, Qianfei Liu, and Chenfan Lian. Pointlanenet: Efficient end-to-end cnns for accurate real-time lane detection. In 2019 IEEE intelligent vehicles symposium (IV), pages 2563–2568. IEEE, 2019.

[186] David C Andrade, Felipe Bueno, Felipe R Franco, Rodrigo Adamshuk Silva, João Henrique Z Neme, Erick Margraf, William T Omoto, Felipe A Farinelli, Angelo M Tusset, Sergio Okida, et al. A novel strategy for road lane detection and tracking based on a vehicle's forward monocular camera. IEEE Transactions on Intelligent Transportation Systems, 20(4):1497–1507, 2018.

[187] Huiqin Zhan and Lin Chen. Lane detection image processing algorithm based on fpga for intelligent vehicle. In 2019 Chinese Automation Congress (CAC), pages 1190–1196. IEEE, 2019.

[188] Mihir Mody, Desappan Kumar, Pramod Swami, Manu Mathew, and Soyeb Nagori. Low cost and power cnn/deep learning solution for automated driving. In 2018 19th International Symposium on Quality Electronic Design (ISQED), pages 432–436. IEEE, 2018.

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

[190] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. IEEE Transactions on Pattern Analysis and Machine Intelligence, 39(6):1137–1149, 2017.

[191] Karam M Abughalieh and Shadi G Alawneh. Predicting pedestrian intention to cross the road. IEEE Access, 8:72558–72569, 2020.

[192] Riadh Ayachi, Mouna Afif, Yahia Said, and Abdessalem Ben Abdelaali. pedestrian detection for advanced driving assisting system: a transfer learning approach. In 2020 5th International Conference on Advanced Technologies for Signal and Image Processing (ATSIP), pages 1–5. IEEE, 2020.

[193] Paulius Tumas, Adam Nowosielski, and Arturas Serackis. Pedestrian detection in severe weather conditions. IEEE Access, 8:62775–62784, 2020.

[194] Redge Melroy Castelino, Gabriel Passos Moreira Pinheiro, Bruno Justino Garcia Praciano, Giovanni Almeida Santos, Lothar Weichenberger, and Rafael Timóteo De Sousa Júnior. Improving the accuracy of pedestrian detection in partially occluded or obstructed scenarios. In 2020 10th International Conference on Advanced Computer Information Technologies (ACIT), pages 834–838. IEEE, 2020.

[195] Wen-Chia Tsai, Kuan-Chou Chen, Jhih-Sheng Lai, and Jiun-In Guo. Front moving object behavior prediction system exploiting deep learning technology for adas applications. In 2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS), pages 1052–1055. IEEE, 2020.

[196] Gianluca Brilli, Paolo Burgio, and Marko Bertogna. Convolutional neural networks on embedded automotive platforms: A qualitative comparison. In 2018 International Conference on High Performance Computing & Simulation (HPCS), pages 496–499. IEEE, 2018.

[197] Mario Vestias and Horacio Neto. Trends of cpu, gpu and fpga for high-performance computing. In 2014 24th International Conference on Field Programmable Logic and Applications (FPL), pages 1–6. IEEE, 2014.

[198] Denis Tomé, Luca Bondi, Luca Baroffio, Stefano Tubaro, Emanuele Plebani, and Danilo Pau. Reduced memory region based deep convolutional neural network detection. In 2016 IEEE 6th International Conference on Consumer Electronics-Berlin (ICCE-Berlin), pages 15–19. IEEE, 2016.

[199] Jie Hua, Ying Shi, Changjun Xie, Hui Zhang, and Jian Zhang. Pedestrian-and vehicle-detection algorithm based on improved aggregated channel features. IEEE Access, 9:25885–25897, 2021.

[200] Ning Zhang, Xin Wei, He Chen, and Wenchao Liu. Fpga implementation for cnn-based optical remote sensing object detection. Electronics, 10(3):282, 2021.

[201] Onur Toker and Suleiman Alsweiss. mmwave radar based approach for pedestrian identification in autonomous vehicles. In 2020 SoutheastCon, pages 1–2. IEEE, 2020.

[202] Xuewen Chen, Huaqing Chen, and Huan Xu. Vehicle detection based on multifeature extraction and recognition adopting rbf neural network on adas system. Complexity, 2020, 2020.

[203] Bartosz Meus, Tomasz Kryjak, and Marek Gorgon. Embedded vision system for pedestrian detection based on hog+ svm and use of motion information implemented in zynq heterogeneous device. In 2017 Signal Processing: Algorithms, Architectures, Arrangements, and Applications (SPA), pages 406–411. IEEE, 2017.

[204] Fong-An Chang, Chia-Chi Tsai, Ching-Kan Tseng, and Jiun-In Guo. Embedded multiple object detection based on deep learning technique for advanced driver assistance system. In 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), pages 172–175. IEEE, 2017.

[205] Riadh Ayachi, Yahia Said, and Abdessalem Ben Abdelaali. Pedestrian detection based on light-weighted separable convolution for advanced driver assistance systems. Neural Processing Letters, 52(3):2655–2668, 2020.

[206] Wen-Chia Tsai, Jhih-Sheng Lai, Kuan-Chou Chen, Vinay M Shivanna, and Jiun-In Guo. A lightweight motional object behavior prediction system harnessing deep learning technology for embedded adas applications. Electronics, 10(6):692, 2021.

[207] Mirza Athar Baig, Muhammad Adeel Pasha, and Shahid Masud. An optimized hardware/software co-design framework for real-time pedestrian detection. In 2020 International Conference on Electrical, Communication, and Computer Engineering (ICECCE), pages 1–6. IEEE, 2020.

[208] Nesrine Abid, Tarek Ouni, Ahmed C Ammari, and Mohamed Abid. Efficient and high-performance pedestrian detection implementation for intelligent vehicles. Multimedia Systems, 28(1):69–84, 2022.

[209] Xiangyu Xu, Jiadi Yu, Yingying Chen, Yanmin Zhu, Shiyou Qian, and Minglu Li. Leveraging audio signals for early recognition of inattentive driving with smartphones. IEEE Transactions on Mobile Computing, 17(7):1553–1567, 2017.

[210] Zheng-Jie Cheng, Lih-Wei Jeng, and Kang Li. Behavioral classification of drivers for driving efficiency related adas using artificial neural network. In 2018 IEEE International Conference on Advanced Manufacturing (ICAM), pages 173–176. IEEE, 2018.

[211] Nengchao Lyu, Chao Deng, Lian Xie, Chaozhong Wu, and Zhihcheng Duan. A field operational test in china: Exploring the effect of an advanced driver assistance system on driving performance and braking behavior. Transportation research part F: traffic psychology and behaviour, 65:730–747, 2019.

[212] Yuan Yuan, Yuwei Lu, and Qi Wang. Adaptive forward vehicle collision warning based on driving behavior. Neurocomputing, 408:64–71, 2020.

[213] Zhiqiang Shen, Zhuang Liu, Jianguo Li, Yu-Gang Jiang, Yurong Chen, and Xiangyang Xue. Dsod: Learning deeply supervised object detectors from scratch. In 2017 IEEE International Conference on Computer Vision (ICCV), pages 1937–1945, 2017.

[214] Ying-Hao Yu, Yi-Siang Ting, Ngaiming Kwok, and Norbert Michael Mayer. High-speed gaze detection using a single fpga for driver assistance systems. Journal of Real-Time Image Processing, 18(3):681–690, 2021.

[215] Peng Ping, Wenhu Qin, Yang Xu, Chiyomi Miyajima, and Kazuya Takeda. Impact of driver behavior on fuel consumption: Classification, evaluation and prediction using machine learning. IEEE Access, 7:78515–78532, 2019.

[216] Bing Zhu, Zhipeng Liu, Jian Zhao, Yizhou Chen, and Weiwen Deng. Driver behavior characteristics identification strategies based on bionic intelligent algorithms. IEEE Transactions on Human-Machine Systems, 48(6):572–581, 2018.

[217] Alexey Kashevnik, Igor Lashkov, and Andrei Gurtov. Methodology and mobile application for driver behavior analysis and accident prevention. IEEE Transactions on Intelligent Transportation Systems, 21(6):2427–2436, 2020.

[218] Qi Deng and Dirk Söffker. Improved driving behaviors prediction based on fuzzy logic-hidden markov model (fl-hmm). In 2018 IEEE Intelligent Vehicles Symposium (IV), pages 2003–2008, 2018.

[219] Bhargava Reddy, Ye-Hoon Kim, Sojung Yun, Chanwon Seo, and Junik Jang. Real-time driver drowsiness detection for embedded system using model compression of deep neural networks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 121–128, 2017.

[220] Miguel García-García, Alice Caplier, and Michèle Rombaut. Sleep deprivation detection for real-time driver monitoring using deep learning. In International conference image analysis and recognition, pages 435–442. Springer, 2018.

[221] Mohammad Shahverdy, Mahmood Fathy, Reza Berangi, and Mohammad Sabokrou. Driver behavior detection and classification using deep convolutional neural networks. Expert Systems with Applications, 149:113240, 2020.

[222] Ziran Wang, Xishun Liao, Chao Wang, David Oswald, Guoyuan Wu, Kanok Boriboonsomsin, Matthew J Barth, Kyungtae Han, BaekGyu Kim, and Prashant Tiwari. Driver behavior modeling using game engine and real vehicle: A learning-based approach. IEEE Transactions on Intelligent Vehicles, 5(4):738–749, 2020.

[223] Vasili Ramanishka, Yi-Ting Chen, Teruhisa Misu, and Kate Saenko. Toward driving scene understanding: A dataset for learning driver behavior and causal reasoning. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, pages 7699–7707, 2018.

[224] Vijay Govindarajan, Katherine Driggs-Campbell, and Ruzena Bajcsy. Affective driver state monitoring for personalized, adaptive adas. In 2018 21st International Conference on Intelligent Transportation Systems (ITSC), pages 1017–1022. IEEE, 2018.

[225] Bing Zhu, Shude Yan, Jian Zhao, and Weiwen Deng. Personalized lane-change assistance system with driver behavior identification. IEEE Transactions on Vehicular Technology, 67(11):10293–10306, 2018.

[226] B. Senouci, H. Rouis, D.S. Han, and E. Bourennane. A hardware skin-segmentation ip for vision based smart adas through an fpga prototyping. In 2017 Ninth International Conference on Ubiquitous and Future Networks (ICUFN), pages 197–199, 2017.

[227] Qingkun Li, Lian Hou, Zhenyuan Wang, Wenjun Wang, Chao Zeng, Quan Yuan, and Bo Cheng. Drivers' visual-distracted take-over performance model and its application on adaptive adjustment of time budget. Accident Analysis & Prevention, 154:106099, 2021.

[228] Agapito Ledezma, Víctor Zamora, Óscar Sipele, M Paz Sesmero, and Araceli Sanchis. Implementing a gaze tracking algorithm for improving advanced driver assistance systems. Electronics, 10(12):1480, 2021.

[229] Iqram Hussain, Seo Young, and Se-Jin Park. Driving-induced neurological biomarkers in an advanced driver-assistance system. Sensors, 21(21):6985, 2021.

[230] Harshitha Bura, Nathan Lin, Naveen Kumar, Sangram Malekar, Sushma Nagaraj, and Kaikai Liu. An edge based smart parking solution using camera networks and deep learning. In 2018 IEEE International Conference on Cognitive Computing (ICCC), pages 17–24. IEEE, 2018.

[231] Paulo RL De Almeida, Luiz S Oliveira, Alceu S Britto Jr, Eunelson J Silva Jr, and Alessandro L Koerich. Pklot–a robust dataset for parking lot classification. Expert Systems with Applications, 42(11):4937–4949, 2015.

[232] Giuseppe Amato, Fabio Carrara, Fabrizio Falchi, Claudio Gennaro, Carlo Meghini, and Claudio Vairo. Deep learning for decentralized parking lot occupancy detection. Expert Systems with Applications, 72:327–334, 2017.

[233] Dongkyu Lee, Jae-Seol Lee, Sinjae Lee, and Seok-Cheol Kee. The real-time implementation for the parking line departure warning system. In 2018 3rd IEEE International Conference on Intelligent Transportation Engineering (ICITE), pages 236–240. IEEE, 2018.

[234] Keras Project Keras-team. Github repository.

[235] Michael G Bechtel, Elise McEllhiney, Minje Kim, and Heechul Yun. Deeppicar: A low-cost deep neural network-based autonomous car. In 2018 IEEE 24th international conference on embedded and real-time computing systems and applications (RTCSA), pages 11–21. IEEE, 2018.

[236] AA Egorchev. System of automatic video stream images evaluation and preprocessing for adas. In IOP Conference Series: Materials Science and Engineering, volume 873, page 012007. IOP Publishing, 2020.

[237] Manu Mathew, Kumar Desappan, Pramod Kumar Swami, and Soyeb Nagori. Sparse, quantized, full frame cnn for low power embedded devices. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, pages 11–19, 2017.

[238] Younkwan Lee, Jihyo Jeon, Jongmin Yu, and Moongu Jeon. Context-aware multitask learning for traffic scene recognition in autonomous vehicles. In 2020 IEEE Intelligent Vehicles Symposium (IV), pages 723–730. IEEE, 2020.

[239] Jindong Zhang, Tong Liu, Xuelong Yin, Xue Wang, Kunpeng Zhang, Jiabin Xu, and Donghui Wang. An improved parking space recognition algorithm based on panoramic vision. Multimedia Tools and Applications, 80(12):18181–18209, 2021.

[240] Lei Wang, Bo Yu, and Chao Chen. Parking area data collection and scenario extraction for the purpose of automatic parking adas function. In IOP Conference Series: Materials Science and Engineering, volume 780, page 032026. IOP Publishing, 2020.

[241] László Kocsány, Anna Barbara Ádám, and Emese Gincsainé Szádeczky-Kardoss. Utilization of collaborative mapping in parking systems. In 2020 23rd International Symposium on Measurement and Control in Robotics (ISMCR), pages 1–6. IEEE, 2020.

[242] Kevin Tirta Wijaya, Luqman Yusuf Bharoto, Andrey Purwanto, and Eniman Yunus Syamsuddin. Vision-based parking assist system with bird-eye surround vision for reverse bay parking maneuver recommendation. In 2020 International Electronics Symposium (IES), pages 102–107. IEEE, 2020.

[243] Markus Heimberger, Jonathan Horgan, Ciarán Hughes, John McDonald, and Senthil Yogamani. Computer vision in automated parking systems: Design, implementation and challenges. Image and Vision Computing, 68:88–101, 2017.

[244] Konrad Haeublein, Wolfgang Brueckner, Steffen Vaas, Sebastian Rachuj, Marc Reichenbach, and Dietmar Fey. Utilizing pynq for accelerating image processing functions in adas applications. In ARCS Workshop 2019; 32nd International Conference on Architecture of Computing Systems, pages 1–8. VDE, 2019.

[245] Yih-Chen Wang, Chao-Wei Yu, Xiu-Ying Lu, and Yen-Lin Chen. Road semantic segmentation and traffic object detection model based on encoder-decoder cnn architecture. In 2022 IEEE International Conference on Consumer Electronics-Taiwan, pages 421–422. IEEE, 2022.

[246] Güner Tatar, Salih Bayar, and İhsan Çiçek. Real-time multi-learning deep neural network on an mpsoc-fpga for intelligent vehicles: Harnessing hardware acceleration with pipeline. IEEE Transactions on Intelligent Vehicles, pages 1–12, 2024.

[247] Mark Nadeski. Bringing machine learning to embedded systems. Texas Instruments, pages 1–5, 2019.

[248] Firas Al-Ali, Thilina Doremure Gamage, Hewa WTS Nanayakkara, Farhad Mehdipour, and Sayan Kumar Ray. Novel casestudy and benchmarking of alexnet for edge ai: From cpu and gpu to fpga. In 2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE), pages 1–4. IEEE, 2020.

[249] Linus Pettersson. Convolutional neural networks on fpga and gpu on the edge: A comparison, 2020.

[250] Yongming Tang, Rongshi Dai, and Yi Xie. Optimization of energy efficiency for fpga-based convolutional neural networks accelerator. In Journal of Physics: Conference Series, volume 1487, page 012028. IOP Publishing, 2020.

[251] E. Szewczyk. Why my gpus beat your asics at crypto mining.

[252] Kh Shahriya Zaman, Mamun Bin Ibne Reaz, Sawal Hamid Md Ali, Ahmad Ashrif A Bakar, and Muhammad Enamul Hoque Chowdhury. Custom hardware architectures for deep learning on portable devices: A review. IEEE Transactions on Neural Networks and Learning Systems, pages 1–21, 2021.

[253] Mikhail Lebedev and Pavel Belecky. A survey of open-source tools for fpga-based inference of artificial neural networks. In 2021 Ivannikov Memorial Workshop (IVMEM), pages 50–56, 2021.

[254] Cort J Willmott, Steven G Ackleson, Robert E Davis, Johannes J Feddema, Katherine M Klink, David R Legates, James O'donnell, and Clinton M Rowe. Statistics for the evaluation and comparison of models. Journal of Geophysical Research: Oceans, 90(C5):8995–9005, 1985.

[255] Aakanshi Gupta, Bharti Suri, Vijay Kumar, and Pragyashree Jain. Extracting rules for vulnerabilities detection with static metrics using machine learning. International Journal of System Assurance Engineering and Management, 12(1):65–76, 2021.

[256] Kazi Zakia Sultana, Vaibhav Anu, and Tai-Yin Chong. Using software metrics for predicting vulnerable classes and methods in java projects: A machine learning approach. Journal of Software: Evolution and Process, 33(3):e2303, 2021.

[257] Danny Hernandez Dario Amodei. Ai and compute.

[258] Steven Woo. Memory is key to future ai and ml performance.

[259] K Siva Sundari and R Narmadha. Design energy efficient shared distributed memory management system on soc's to improve memory performance. Applied Nanoscience, pages 1–11, 2022.

[260] Yaman Umuroglu, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. Finn: A framework for fast, scalable binarized neural network inference. In Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '17, pages 65–74. ACM, 2017.

[261] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In European conference on computer vision, pages 525–542. Springer, 2016.

[262] Wonyong Sung, Sungho Shin, and Kyuyeon Hwang. Resiliency of deep neural networks under quantization. arXiv preprint arXiv:1511.06488, 2015.

[263] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. arXiv preprint arXiv:1602.02830, 2016.

[264] Codementor Blog. Machine learning: How to build scalable machine learning models.

[265] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan. Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices. In 2021 IEEE International Symposium on Workload Characterization (IISWC), pages 226–237, 2021.

[266] Yan Li and Zhiheng Huang. Basics and applications of ai in adas and autonomous vehicles. Advanced Driver Assistance Systems and Autonomous Vehicles, pages 17–48, 2022.

[267] Zhuoer Li, Edgar Lemaire, Nassim Abderrahmane, Sébastien Bilavarn, and Benoit Miramond. Efficiency analysis of artificial vs. spiking neural networks on fpgas. Journal of Systems Architecture, 133:102765, 2022.

[268] asicNorth. Asic vs. fpga: What's the difference?

[269] Satyajit Bora and Roy Paily. Design and implementation of adaptive binary divider for fixed-point and floating-point numbers. Circuits, Systems, and Signal Processing, 41(2):1131–1145, 2022.

[270] Ahmad Shawahna, Sadiq M Sait, Aiman El-Maleh, and Irfan Ahmad. Fxp-qnet: A post-training quantizer for the design of mixed low-precision dnns with dynamic fixed-point representation. IEEE Access, 2022.

[271] Don Lahiru Nirmal Hettiarachchi and Eric J Balster. Fixed-point processing of the sar back-projection algorithm on fpga. IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing, 14:10889–10902, 2021.

[272] Naveen Suda, Vikas Chandra, Ganesh Dasika, Abinash Mohanty, Yufei Ma, Sarma Vrudhula, Jae-sun Seo, and Yu Cao. Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks. In Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, pages 16–25, 2016.

[273] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In International conference on machine learning, pages 1737–1746. PMLR, 2015.

[274] Kevin Kiningham, Michael Graczyk, Athul Ramkumar, and SCPD Stanford. Design and analysis of a hardware cnn accelerator. Small, 27:6, 2016.

[275] Xuechao Wei, Cody Hao Yu, Peng Zhang, Youxiang Chen, Yuxin Wang, Han Hu, Yun Liang, and Jason Cong. Automated systolic array architecture synthesis for high throughput cnn inference on fpgas. In Proceedings of the 54th Annual Design Automation Conference 2017, pages 1–6, 2017.

[276] Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays, pages 161–170, 2015.

[277] Vincent Vanhoucke, Andrew W. Senior, and Mark Z. Mao. Improving the speed of neural networks on cpus. 2011.

[278] Maurizio Capra, Beatrice Bussolino, Alberto Marchisio, Guido Masera, Maurizio Martina, and Muhammad Shafique. Hardware and software optimizations for accelerating deep neural networks: Survey of current trends, challenges, and the road ahead. IEEE Access, 8:225134–225180, 2020.

[279] Arish Sateesan, Sharad Sinha, Smitha KG, and AP Vinod. A survey of algorithmic and hardware optimization techniques for vision convolutional neural networks on fpgas. Neural Processing Letters, 53(3):2331–2377, 2021.

[280] Yuhao Zhu, Matthew Mattina, and Paul Whatmough. Mobile machine learning hardware at arm: a systems-on-chip (soc) perspective. arXiv preprint arXiv:1801.06274, 2018.

[281] Suejb Memeti, Sabri Pllana, Alécio Binotto, Joanna Kołodziej, and Ivona Brandic. Using meta-heuristics and machine learning for software optimization of parallel computing systems: a systematic literature review. Computing, 101(8):893–936, 2019.

[282] Matheus S Serpa, Arthur M Krause, Eduardo HM Cruz, Philippe Olivier Alexandre Navaux, Marcelo Pasin, and Pascal Felber. Optimizing machine learning algorithms on multi-core and many-core architectures using thread and data mapping. In 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pages 329–333. IEEE, 2018.

[283] Dongjoo Shin, Jinmook Lee, Jinsu Lee, Juhyoung Lee, and Hoi-Jun Yoo. Dnpu: An energy-efficient deep-learning processor with heterogeneous multi-core architecture. IEEE Micro, 38(5):85–93, 2018.

[284] Dhiraj Kalamkar, Evangelos Georganas, Sudarshan Srinivasan, Jianping Chen, Mikhail Shiryaev, and Alexander Heinecke. Optimizing deep learning recommender systems training on cpu cluster architectures. In SC20: International Conference for High Performance Computing, Networking, Storage and Analysis, pages 1–15. IEEE, 2020.

[285] Dheevatsa Mudigere, Yuchen Hao, Jianyu Huang, Zhihao Jia, Andrew Tulloch, Srinivas Sridharan, Xing Liu, Mustafa Ozdal, Jade Nie, Jongsoo Park, et al. Software-hardware co-design for fast and scalable training of deep learning recommendation models. arXiv preprint arXiv:2104.05158, 2021.

[286] Ross Graham. Discourse analysis of academic debate of ethics for agi. AI & SOCIETY, pages 1–14, 2021.

[287] Shiliang Sun, Zehui Cao, Han Zhu, and Jing Zhao. A survey of optimization methods from a machine learning perspective. IEEE transactions on cybernetics, 50(8):3668–3681, 2019.

[288] Sebastian Ruder. An overview of gradient descent optimization algorithms. arXiv preprint arXiv:1609.04747, 2016.

[289] Joshua Alspector, Ronny Meir, B Yuhas, Anthony Jayakumar, and D Lippe. A parallel gradient descent method for learning in analog vlsi neural networks. Advances in neural information processing systems, 5, 1992.

[290] Jorge Nocedal and Stephen J. Wright. Numerical Optimization. Springer, New York, NY, USA, 2e edition, 2006.

[291] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11):2278–2324, 1998.

[292] Rie Johnson and Tong Zhang. Accelerating stochastic gradient descent using predictive variance reduction. Advances in neural information processing systems, 26, 2013.

[293] Arkadi Nemirovski, Anatoli Juditsky, Guanghui Lan, and Alexander Shapiro. Robust stochastic approximation approach to stochastic programming. SIAM Journal on optimization, 19(4):1574–1609, 2009.

[294] Alekh Agarwal, Martin J Wainwright, Peter Bartlett, and Pradeep Ravikumar. Information-theoretic lower bounds on the oracle complexity of convex optimization. Advances in Neural Information Processing Systems, 22, 2009.

[295] Nicolas Roux, Mark Schmidt, and Francis Bach. A stochastic gradient method with an exponential convergence _rate for finite training sets. Advances in neural information processing systems, 25, 2012.

[296] Matthew D Zeiler. Adadelta: an adaptive learning rate method. arXiv preprint arXiv:1212.5701, 2012.

[297] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980, 2014.

[298] Geoffrey Hinton, Nitish Srivastava, and Kevin Swersky. Neural networks for machine learning lecture 6a overview of mini-batch gradient descent. Cited on, 14(8):2, 2012.

[299] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. Journal of machine learning research, 12(7), 2011.

[300] Ning Qian. On the momentum term in gradient descent learning algorithms. Neural networks, 12(1):145–151, 1999.

[301] Herbert Robbins and Sutton Monro. A stochastic approximation method. The annals of mathematical statistics, pages 400–407, 1951.

[302] Jeffrey Larson, Matt Menickelly, and Stefan M Wild. Derivative-free optimization methods. Acta Numerica, 28:287–404, 2019.

[303] Sambhav Jain, Albert Gural, Michael Wu, and Chris Dick. Trained quantization thresholds for accurate and efficient fixed-point inference of deep neural networks. Proceedings of Machine Learning and Systems, 2:112–128, 2020.

[304] Masoud Aliramezani, Charles Robert Koch, and Mahdi Shahbakhti. Modeling, diagnostics, optimization, and control of internal combustion engines via modern machine learning techniques: A review and future directions. Progress in Energy and Combustion Science, 88:100967, 2022.

[305] Congliang Chen, Li Shen, Wei Liu, and Zhi-Quan Luo. Efficient-adam: Communication-efficient distributed adam with complexity analysis. arXiv preprint arXiv:2205.14473, 2022.

[306] Pavel Andreev and Alexander Fritzler. Quantization of generative adversarial networks for efficient inference: a methodological study. In 2022 26th International Conference on Pattern Recognition (ICPR), pages 2179–2185. IEEE, 2022.

[307] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. Advances in neural information processing systems, 28, 2015.

[308] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. arXiv preprint arXiv:1510.00149, 2015.

[309] Bratislav Predić, Uroš Vukić, Muzafer Saračević, Darjan Karabašević, and Dragiša Stanujkić. The possibility of combining and implementing deep neural network compression methods. Axioms, 11(5):229, 2022.

[310] Guangli Li, Xiu Ma, Xueying Wang, Hengshan Yue, Jiansong Li, Lei Liu, Xiaobing Feng, and Jingling Xue. Optimizing deep neural networks on intelligent edge accelerators via flexible-rate filter pruning. Journal of Systems Architecture, 124:102431, 2022.

[311] Yidan Zhang, Guangjin Wang, Taibo Yang, Tianfeng Pang, Zhenan He, and Jiancheng Lv. Compression of deep neural networks: bridging the gap between conventional-based pruning and evolutionary approach. Neural Computing and Applications, pages 1–22, 2022.

[312] Chen Wu, Mingyu Wang, Xinyuan Chu, Kun Wang, and Lei He. Low-precision floating-point arithmetic for high-performance fpga-based cnn acceleration. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 15(1):1–21, 2021.

[313] Chenglong Huang, Puguang Liu, and Liang Fang. Mxqn: mixed quantization for reducing bit-width of weights and activations in deep convolutional neural networks. Applied Intelligence, 51(7):4561–4574, 2021.

[314] Di Liu, Hao Kong, Xiangzhong Luo, Weichen Liu, and Ravi Subramaniam. Bringing ai to edge: From deep learning's perspective. Neurocomputing, 485(C):297–320, May 2022.

[315] Manish Gupta and Puneet Agrawal. Compression of deep learning models for text: A survey. ACM Transactions on Knowledge Discovery from Data (TKDD), 16(4):1–55, 2022.

[316] Bo Chen, Ali Bakhshi, Gustavo Batista, Brian Ng, and Tat-Jun Chin. Update compression for deep neural networks on the edge. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 3076–3086, 2022.

[317] Satish S Bhairannawar. Efficient medical image enhancement technique using transform hsv space and adaptive histogram equalization. In Soft Computing Based Medical Image Analysis, pages 51–60. Elsevier, 2018.

[318] Murad Qasaimeh, Kristof Denolf, Alireza Khodamoradi, Michaela Blott, Jack Lo, Lisa Halder, Kees Vissers, Joseph Zambreno, and Phillip H Jones. Benchmarking vision kernels and neural network inference accelerators on embedded platforms. Journal of Systems Architecture, 113:101896, 2021.

[319] Xin Jiang. Retracted: Human tracking of track and field athletes based on fpga and computer vision, 2021.

[320] Zynq 7000 SoC — xilinx.com. https://www.xilinx.com/products/silicon-devices/soc/zynq-7000.html. [Accessed 29-03-2024].

[321] Sam Bobrowicz. Vivado Library - Digilent Reference — digilent.com. https://digilent.com/reference/vivado:library. [Accessed 29-03-2024].

[322] Sagar Eetha, Sonali Agrawal, and Srikanth Neelam. Zynq fpga based system design for video surveillance with sobel edge detection. In 2018 IEEE International Symposium on Smart Electronic Systems (iSES)(Formerly iNiS), pages 76–79. IEEE, 2018.

[323] Abdelkader Ben Amara, Edwige Pissaloux, and Mohamed Atri. Sobel edge detection system design and integration on an fpga based hd video streaming architecture. In 2016 11th International Design & Test Symposium (IDT), pages 160–164. IEEE, 2016.

[324] Tarun Agarwal. What is FIR Filter? - FIR Filters for Digital Signal Processing — elprocus.com. https://www.elprocus.com/fir-filter-for-digital-signal-processing/. [Accessed 29-03-2024].

[325] Mountassar Maamoun, Adnane Hassani, Samir Dahmani, Hocine Ait Saadi, Ghania Zerari, Noureddine Chabini, and Rachid Beguenane. Efficient fpga based architecture for high-order fir filtering using simultaneous dsp and lut reduced utilization. IET Circuits, Devices & Systems, 15(5):475–484, 2021.

[326] Xue Xian Zheng, Jun Yang, Si Yuan Yang, Wangqian Chen, Li Ying Huang, and Xiu Yin Zhang. Synthesis of linear-phase fir filters with a complex exponential impulse response. IEEE Transactions on Signal Processing, 69:6101–6115, 2021.

[327] Güner Tatar, Salih Bayar, and İhsan Çiçek. Fpga design of a high-resolution fir band-pass filter by using labview environment. Avrupa Bilim ve Teknoloji Dergisi, (29):273–277, 2021.

[328] Güner Tatar, İhsan Çiçek, and Salih Bayar. Fpga design of a fourth order elliptic iir band-pass filter using labview. Avrupa Bilim ve Teknoloji Dergisi, (26):122–127, 2021.

[329] Peter Isza. TFilter - Free online FIR filter design — t-filter.engineerjs.com. http://t-filter.engineerjs.com/. [Accessed 29-03-2024].

[330] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 11(3):1–23, 2018.

[331] finn/notebooks at main · Xilinx/finn — github.com. https://github.com/Xilinx/finn/tree/main/notebooks. [Accessed 29-03-2024].

[332] Alessandro Pappalardo. Xilinx/brevitas, 2023.

[333] Lei Mao. Quantization for Neural Networks — leimao.github.io. https://leimao.github.io/article/Neural-Networks-Quantization/. [Accessed 29-03-2024].

[334] ONNX Runtime developers. Onnx runtime. https://onnxruntime.ai/, 2021. Version: x.y.z.

[335] Michaela Blott, Thomas B Preußer, Nicholas J Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. Finn-r: An end-to-end deep-learning framework for fast exploration of quantized neural networks. ACM Transactions on Reconfigurable Technology and Systems (TRETS), 11(3):1–23, 2018.

[336] BCEWithLogitsLoss – pytorch 2.2 documentation. Accessed 29-03-2024.

[337] Jian-Hao Luo and Jianxin Wu. Autopruner: An end-to-end trainable filter pruning method for efficient deep model inference. Pattern Recognition, 107:107461, 2020.

[338] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. Neurocomputing, 461:370–403, 2021.

[339] Xilinx. Vitis-ai/model_zoo at 2.5. https://github.com/Xilinx/Vitis-AI/tree/2.5/model_zoo, 2024. Accessed on 02-04-2024.

[340] François Chollet. Xception: Deep learning with depthwise separable convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 1251–1258, 2017.

[341] Kevis-Kokitsi Maninis, Ilija Radosavovic, and Iasonas Kokkinos. Attentive single-tasking of multiple tasks. pages 1851–1860, 2019.

[342] Menelaos Kanakis, David Bruggemann, Suman Saha, Stamatios Georgoulis, Anton Obukhov, and Luc Van Gool. Reparameterizing convolutions for incremental multi-task learning without task interference. In Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part XX 16, pages 689–707. Springer, 2020.

[343] Mário P Véstias, Rui P Duarte, José T de Sousa, and Horácio C Neto. A fast and scalable architecture to run convolutional neural networks in low density fpgas. Microprocessors and Microsystems, 77:103136, 2020.

[344] AMD. Vitis AI Quantizer User Guide. AMD, 2024. Accessed on 02-04-2024.

[345] Pierre-Emmanuel Novac, Ghouthi Boukli Hacene, Alain Pegatoquet, Benoit Miramond, and Vincent Gripon. Quantization and deployment of deep neural networks on microcontrollers. Sensors, 21(9):2984, 2021.

[346] Yoshua Bengio, Nicholas Léonard, and Aaron C. Courville. Estimating or propagating gradients through stochastic neurons for conditional computation. ArXiv, abs/1308.3432, 2013.

[347] Steven K Esser, Jeffrey L McKinstry, Deepika Bablani, Rathinakumar Appuswamy, and Dharmendra S Modha. Learned step size quantization. arXiv preprint arXiv:1902.08153, 2019.

[348] Fisher Yu, Haofeng Chen, Xin Wang, Wenqi Xian, Yingying Chen, Fangchen Liu, Vashisht Madhavan, and Trevor Darrell. Bdd100k: A diverse driving dataset for heterogeneous multitask learning. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2636–2645, 2020.

[349] Pei Sun, Henrik Kretzschmar, Xerxes Dotiwalla, Aurelien Chouard, Vijaysai Patnaik, Paul Tsui, James Guo, Yin Zhou, Yuning Chai, Benjamin Caine, et al. Scalability in perception for autonomous driving: Waymo open dataset. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition, pages 2446–2454, 2020.

[350] Jannik Fritsch, Tobias Kuehnl, and Andreas Geiger. A new performance measure and evaluation benchmark for road detection algorithms. In 16th International IEEE Conference on Intelligent Transportation Systems (ITSC 2013), pages 1693–1700. IEEE, 2013.

[351] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 3213–3223, 2016.

[352] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition, pages 248–255. Ieee, 2009.

[353] Maxim Berman, Amal Rannen Triki, and Matthew B Blaschko. The lovász-softmax loss: A tractable surrogate for the optimization of the intersection-over-union measure in neural networks. In Proceedings of the IEEE conference on computer vision and pattern recognition, pages 4413–4421, 2018.

[354] David Eigen, Christian Puhrsch, and Rob Fergus. Depth map prediction from a single image using a multi-scale deep network. Advances in neural information processing systems, 27, 2014.

[355] Akira Kojima and Yuya Osawa. Design and implementation of autonomous driving robot car using soc fpga. In 2019 International Conference on Field-Programmable Technology (ICFPT), pages 441–444. IEEE, 2019.

Appendix A: Publications Within the Scope of the Thesis

SCI and/or SCI-E Index Journals

Article Name	Year	Published In
Recent Advances in Machine Learning based Advanced Driver Assistance System Applications	2024	Microprocessors and Microsystems
Real-Time Multi-Learning Deep Neural Network on an MPSoC-FPGA for Intelligent Vehicles Harnessing Hardware Acceleration with Pipeline	2024	IEEE Transactions on Intelligent Vehicles
Real-Time Multi-Task A/DAS Implementation on Reconfigurable Heterogeneous MPSoC Architecture	2023	IEEE Access

Turkish Index Journals

Article Name	Year	Published In
FPGA Design of a High-Resolution FIR Band-Pass Filter by Using LabVIEW Environment	2021	European Journal of Science and Technology, pages 273&277
FPGA design of a fourth order elliptic IIR band-pass filter using LabVIEW	2021	European Journal of Science and Technology, pages 122&127

International Conference Proceedings

Article Name	Year	Published In
Performance Evaluation of Real-Time Video Processing Edge Detection on Various Platforms	2023	IEEE 17th International Conference on Application of Information and Communication Technologies
Real-Time Hardware Acceleration of Low Precision Quantized Custom Neural Network Model on ZYNQ SoC	2023	5th International Congress on Human-Computer Interaction, Optimization and Robotic Applications
Performance Evaluation of Low-Precision Quantized LeNet and ConvNet Neural Networks	2022	International Conference on Innovations in Intelligent Systems and Applications

Article Name	Year	Published In
Hardware Acceleration of FIR Filter Implementation on ZYNQ SoC	2022	IEEE 16th International Conference on Application of Information and Communication Technologies
FPGA based fault distance detection and positioning of underground energy cable by using GSM/GPRS	2019	International Symposium on Advanced Electrical and Communication Technologies
FPGA Based Step Motor Control For Solar Panels	2019	IEEE 13th International Conference on Application of Information and Communication Technologies
FPGA based Bluetooth Controlled Land Vehicle	2018	International Symposium on Advanced Electrical and Communication Technologies