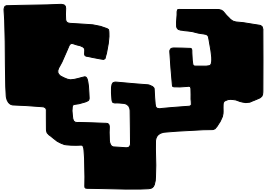
eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

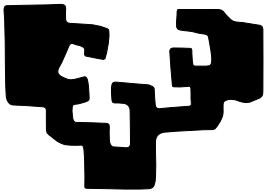
OPTIMIZING DEEP NEURAL NETWORK DEPLOYMENT FOR INTELLIGENT SECURITY VIDEO ANALYTICS

by:
Unai Elordi Hidalgo

Supervised by:
Dr. Ignacio Arganda Carreras
&
Dr. Luis Unzueta Irurtia

Donostia – San Sebastian, May 31, 2023

eman ta zabal zazu



Universidad del País Vasco
Euskal Herriko Unibertsitatea

DEPARTMENT OF COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE

OPTIMIZING DEEP NEURAL NETWORK DEPLOYMENT FOR INTELLIGENT SECURITY VIDEO ANALYTICS

by:
Unai Elordi Hidalgo

Supervised by:
Dr. Ignacio Arganda Carreras
&
Dr. Luis Unzueta Irurtia

Donostia – San Sebastian, May 31, 2023

Abstract

Intelligent Security Video Analytics (ISVA) powered by Artificial Intelligence (AI) offer a solution to the limitations of traditional video surveillance, limitations such as manual monitoring inefficiencies and the inability to detect security threats in real-time. Deep Neural Networks (DNNs) have proven to be highly effective in video analytics, but their deployment in current intelligent system infrastructures is challenging due to their high computational power requirements. These networks are complex inter-connected neuron graphs with a large number of parameters and arithmetic operations, resulting in processing bottlenecks. Moreover, DNN-based vision applications involve processing pipelines with multiple DNN inference tasks and data pre- and post-processing operations, leading to increased processing overhead. Although several DNN optimization techniques have been proposed, new end-to-end pipeline optimization strategies are necessary as more complex DNN-based intelligent systems are developed.

To overcome these DNN computing limitations, new AI accelerators have emerged (Google's TPU, Intel's VPU, etc. that we can generically call "xPU", and also improved CPU, GPU and FPGA architectures), designed to be energy-efficient and to enhance the performance of DNN inference. However, deploying these devices effectively requires significant expertise and analysis. Additionally, the cost of these devices may not be justified unless they are utilized efficiently.

ISVA systems can be integrated into heterogeneous, decentralized, and distributed computing platforms, such as cloud or fog environments, and offline edge devices or in the Internet of Things (IoT) context. These infrastructures require a comprehensive analysis of the optimal deployment

strategy due to their diversity, including multiple devices, application architectures, and runtimes. Unfortunately, these runtimes and devices are not designed for DNN inference, and there is a lack of support for AI accelerators in such diverse computing platforms.

This thesis aims to enhance the deployment of DNN-based vision applications in intelligent system infrastructures through the latest advances in DNN optimization techniques and tailored deployment strategies for Computer Vision (CV). The research aims to improve the efficiency and effectiveness of ISVA, enabling seamless integration into intelligent system infrastructures and paving the way for smarter IoT systems. By optimizing DNN and CV techniques and deployment strategies, the thesis seeks to overcome the challenges associated with the computational requirements of DNN-based vision applications.

To achieve the objectives of enhancing the deployment of DNN-based vision applications in intelligent system infrastructures, this study proposes different deployment strategies of DNNs for ISVA in different environments. These environments include (1) serverless cloud environments for large-scale and highly variable computing workloads, (2) heterogeneous IoT platforms, and (3) edge devices with low-computational resources and demanding workloads.

The first strategy provides a comprehensive performance evaluation of DNN inference in serverless cloud environments and multi-DNN deployment strategies in the context of video surveillance. The second strategy analyzes the optimal deployment in heterogeneous IoT environments of multi-DNN-based face recognition algorithms including face quality assessment, anti-spoofing, face verification, and managing secure biometric data. Finally, the third strategy optimizes an end-to-end multi-DNN deployment pipeline for aircraft cabin readiness verification.

Their outcomes reveal that the deployment of DNN-based solutions for ISVA can benefit from on-site benchmarking and knowledge-based procedures to leverage the computing capabilities of the targeted computing platforms in each use case.

Resumen

La vídeo analítica inteligente para la seguridad actualmente aplica algoritmos de Inteligencia Artificial (IA) que permiten dar solución a las limitaciones de la vídeo vigilancia tradicional, que hasta ahora se basa meramente en la monitorización manual y con una capacidad muy limitada para detectar amenazas de seguridad en tiempo real.

Las redes neuronales profundas (Deep Neural Networks, DNNs, en inglés) han demostrado ser altamente efectivas para el análisis de vídeo, pero su implementación en las infraestructuras actuales de sistemas inteligentes implican un reto importante debido a sus altos requisitos computacionales. Estas redes son grafos complejos de neuronas interconectadas con un gran número de parámetros y operaciones aritméticas, lo que resulta en cuellos de botella en el procesamiento. Además, las aplicaciones de visión basadas en DNNs involucran múltiples operaciones de inferencia de DNN y de pre- y post-procesamiento de datos, lo que lleva a una mayor sobrecarga de procesamiento. Aunque se han propuesto numerosas técnicas de optimización de DNNs, se necesitan nuevas estrategias de optimización en aplicaciones de extremo-a-extremo (end-to-end) basados en sistemas inteligentes de visión basados en DNNs.

Con objetivo de dar respuesta a los requisitos computacionales de los DNNs, los nuevos aceleradores de IA han ido apareciendo en el mercado como (Las TPUs de Google, VPUs de Intel, etc., que se pueden denominar de manera generica como genérica "xPU", así como arquitecturas mejoradas de CPU, GPU y FPGA), diseñados para ser energéticamente eficientes y con un incremento de rendimiento muy considerable para la inferencia DNN. Sin embargo, desplegar de forma eficiente las aplicaciones basadas en DNNs

dentro de estos dispositivos requiere de un expertise y know-how significativos. Además, el coste económico de estos dispositivos puede no estar justificado a menos que se utilicen eficientemente. Los sistemas de ISVA pueden estar integrados en plataformas de computación heterogéneas, descentralizadas y distribuidas como entornos en la nube (cloud), la niebla (fog) y dispositivos en extremo (edge) sin conexión o en el contexto de Internet de las Cosas (Internet of Things, IoT, en inglés). Estas plataformas requieren de un análisis exhaustivo en decidir la estrategia óptima de despliegue debido a su diversidad, incluyendo múltiples dispositivos, arquitecturas de aplicaciones y entornos virtualizados de ejecución. Desafortunadamente, estos entornos virtualizados y sus dispositivos hardware asociados no están diseñados para la inferencia DNN, y por lo tanto, no existe un soporte para desplegar estos sistemas entre la diversas opciones de los aceleradores de IA.

Esta tesis tiene como objetivo optimizar el despliegue de aplicaciones de visión basadas en DNN dentro de las infraestructuras de sistemas inteligentes teniendo en cuenta los últimos avances en técnicas de optimización y estrategias de despliegue personalizadas de DNNs para visión por computador (Computer Vision, CV, en inglés). La investigación de este trabajo tiene como objetivo mejorar la eficiencia y la eficacia de los sistemas de vídeo analítica, habilitando un proceso de despliegue optimo para los sistemas inteligentes y entornos IoT. Basandose en estas técnicas y estrategias de despliegue de DNNs para CV, esta tesis pretende dar solucion a los desafíos asociados a los requisitos computacionales de las aplicaciones de visión artificial basadas en DNNs.

Para lograr estos objetivos, este estudio propone diferentes estrategias de despliegue de DNN para análisis de videoen diferentes entornos de ejecución. Estos entornos incluyen (1) entornos en la nube serverless para delegar procesos a gran escala con una gran variabildad, (2) plataformas IoT heterogéneas y (3) dispositivos edge con recursos computacionales limitados con procesos computacionales exigentes. La primera estrategia proporciona una evaluación integral del rendimiento de la inferencia

de DNN en entornos en la nube serverless y estrategias de despliegue de múltiples DNN en el contexto de la video analítica. La segunda estrategia analiza el despliegue óptimo de algoritmos de reconocimiento facial basados en múltiples DNN en entornos de IoT heterogéneos, que incluyen evaluación de calidad facial, anti-suplantación de identidad, verificación facial y gestión de datos biométricos seguros. Finalmente, la tercera estrategia optimiza un flujo proceso constante de vídeo analítica desplegado con múltiples DNNs ejecutándose en dispositivos de bajos recursos para solventar la problemática de la correcta posición del equipaje de mano en la cabina del avión Los resultados de esta tesis revelan que el despliegue de soluciones basadas en DNN para ISVA pueden beneficiarse de evaluaciones comparativas in-situ y de procedimientos basados en el conocimiento para aprovechar las capacidades computacionales de las plataformas en cada caso de uso.

Laburpena

Adimen Artifizialean (Artificial Intelligence, AI, ingelesez) oinarritutako segurtasunerako-bideo-analisi adimendunak ((Intelligent security video analytics, ISVA, ingelesez) bideo-zaintza tradizionalaren mugei irtenbidea eman diezaiokete. Bideo analisi sistema tradizional hauek orain arte eraginkortasun gutxiko begi bistako monitorizazio hutsa egiten dute eta arazoak dituzte segurtasun mehatxuen aurrean behar bezala erreakzionatzeko.

Sare neuronal sakonak (Deep Neural Networks, DNNs, ingelesez) oso eraginkorrak direla frogatu da bideoaren analisiaren munduan, baina egungo sistema adimendunetako azpiegituretan ezartzea benetan zaila izan daiteke, batez ere, gaitasun konputazional handia eskatzen dutelako DNNak. Sare sakon hauek beraien artean interkonektatuta dauden neuronen grafo konplexuak dira, parametro eta eragiketa aritmetiko ugari dituztenak, eta horrek prozesamendu geldo bat ekarri dezake. Horretaz gain, DNNetan oinarritutako ikusmen aplikazioek datuen pre eta post prozesamendua gehitu behar diote DNN inferentziari eta beraz, honek prozesamendu guztiaren denbora gehitzen du. DNNak optimizatzeko teknika ugari proposatu diren arren, muturretik-muturrerainoko optimizazio estrategia berriak behar dira DNNetan oinarritutako sistema adimendun konplexuagoak garatzen diren heinean.

DNN konputazio-muga horiei aurre egiteko, AI azeleragailu berriak (Googleren TPUa, Intelen VPUa, etab., orokorrean "xPU" dei diezaiokeguna, baita CPU, GPU eta FPGA arkitekturak ere) sortzen ari dira, energetikoki eraginkorrak izateko eta DNN inferentzia-errendimendu hobeagoak emateko diseinatuta baitaude. Hala ere, gailu hauek modu eraginkorrean ezartzeak analisi sakona behar du. Era berean, baliteke gailu horien kostua ez justifikatua egotea eraginkortasunez erabiltzen ez badira.

ISVA sistemak konputazio-plataforma heterogeneo, deszentralizatu eta banatuetan txertatu daitezke, hala nola, hodei (cloud) edo laino (fogo) inguruneetan eta ertzeko gailuetan (edge) edo Gauzen Interneteko (Internet of Things, IoT, ingelesez) testuinguruan. Plataforma hauek inplementazio-estrategia optimoaren azterketa zabala behar dute haien aniztasuna dela eta, hainbat gailu, aplikazio-arkitektura eta exekuzio-denborak kontuan hartu behar baitira. Zoritxarrez, birtualizatutako exekuzio inguru eta gailu hauek ez daude DNN inferentziarako diseinatuta eta AI azeleragailuentzako integrazio falta da dago konputazio-plataforma anitz hauek sustatzeko.

Tesi honek DNNetan oinarritutako ikusmen aplikazioen hedapena hobetu nahi du sistema adimendunen azpiegituretan, optimizazio-tekniketan eta DNNak ikusmen artifizialerako (Computer Vision, CV, ingelesez) pertsonalizatutako hedapen-estrategien azken aurrerapenen bidez. Ikerketak ISVAren adimen artifizialeko algoritmoen eraginkortasuna hobetzea du helburu, sistema adimendunen azpiegituretan integrazio ahalbidetuz eta IoT sistema adimentsuagoetarako bidea irekiz.

Sistema adimendunen azpiegituretan DNNetan oinarritutako ikusmen aplikazioen hedapena hobetzeko helburuak lortzeko, ikerketa honek, DNNen gaitasuna hobetzeko estrategia desberdinak proposatzen ditu ISVAko ingurune ezberdinetan. Ingurune horien artean daude (1) zerbitzaririk gabeko hodei-inguruneak eskala handiko eta oso aldakorreko konputazio-lan kargak egiteko (serverless), (2) IoT plataforma heterogeneoak eta (3) konputazio baliabide mugatuak eta lan-karga mugatuak jaso ditzaketen konputazio inguruneak.

Lehenengo estrategiak DNN inferentziaren errendimenduaren ebaluazio integrala eskaintzen du zerbitzaririk gabeko hodei-inguruneetan eta DNN anitzeko estrategiak bideo-zaintzaren testuinguruan. Bigarren estrategiak DNN anitzetan oinarritutako aurpegi-ezagutza algoritmoen esparruan aritzen da IoT ingurune heterogeneoetan. Azkenik, hirugarren estrategiak DNN anitzeko bideo analisis pipelineak optimizatzen ditu du hegazkinen kabinaren segurtasuna bermatzeko.

Tesi honen emaitzak agerian uzten ditu DNNn oinarritutako soluzio optimizatuak beharrezkoak direla ISVArako eta ezagutzan oinarritutako prozedurei etekina atera diezaiekeela erabilera kasu bakoitzean.

Acknowledgements

First and foremost, I would like to express my most sincere gratitude to my supervisors Luis Unzueta (Vicomtech) and Ignacio Arganda Carreras (Euskal Herriko Unibertsitatea). Your patience, empathy, courage, and constant supervision have created an efficient environment in which I have been able to thrive as a researcher. Thank you both for your support and guidance.

I am so grateful to be a part of Intelligent Transport System department and Video Surveillance and Security department. Thanks to Jorge Garcia and Oihana Otaegui for helping me to find the most appropriate projects and resources to carry out this researching work.

Special thanks to Julián Florez, Edurne Loyarte, and Jorge Posada for their confidence in me and for believing in the feasibility of this PhD dissertation thesis like I did. Vicomtech has provide me with a comfortable environment and resources to carry out this dissertation thesis.

Special thanks to Nerea Aranjuelo, Jon Goenetxea and Jose Luis Apellaniz for contributing to this research work. Also, thanks to Carlos Toro for his unconditional willingness and support during the final stages of this thesis.

I would like to thank my parents and my brother, for their love and unconditional support; for always believing in me and encouraging me to pursue my dreams.

Last but not least, thank you Kattalin for being there in the bad and good moments. Your emotional support, patience and love are essential in my life. Lea, welcome to the family, we have a long long way to live together. The following pages of the dissertation are dedicated to you.

Eskerrik asko

Unai Elordi Hidalgo

May 2023

Contents

List of Figures

List of Tables

Part I

Introduction

CHAPTER

1

Introduction

1.1 Motivation

Intelligent security video analytics (ISVA) can automatically and accurately analyze detailed situational awareness from video surveillance images [5]. Recent advances in Artificial Intelligence (AI) and computer vision have greatly enhanced the cognitive capabilities of ISVA systems. In particular, the outstanding prediction capabilities of Deep Neural Networks (DNNs) have driven the development of diverse DNN-based end-to-end vision applications for video analytics in the Machine Learning (ML) field, including face recognition, potential security threat detection, person and object tracking [6].

The integration of DNN-based end-to-end vision applications in ISVA systems can significantly improve the efficiency of security operations. More specifically, these applications can alleviate the workload of security personnel who often face time-consuming duties that are prone to human error.

To effectively integrate these DNN-based applications within an ISVA system, it is necessary to follow a comprehensive deployment process that includes data acquisition, training the DNN model, optimizing the performance of the model, and the deployment in AI workloads. There are currently practical guidelines such as Machine Learning Operations (MLOps) that can help to optimize the deployment process and

ensure success of the integration [7]. Also, in order to put these applications into production requires the assessment of the technology readiness level for machine learning applications (MLTRL) [8]. This structured evaluation method enables technology developers and managers by providing clear descriptions of the current technology status. MLTRL contain 10 levels to define the technology maturity level, from the designing basic principles for a ML problem (0) to ready for deployment in production (9). Research on MLOps is in its early stages [9] and there is significant room for improvement in standardizing these practical guidelines to efficiently ensure the final MLTRL levels. MLOps is based on a life cycle which is clearly divided into two phases: ML (for learning and training phase) and the Ops (for DNN deployment for AI workloads).

This thesis is focused on dealing with the challenges and the research work for the Ops phase. More specifically, this thesis aims to create deployment strategies that are (1) adapted to the decentralized, distributed and heterogeneous environments of intelligent system infrastructures; (2) tailored to the hardware device architectures and software runtime / framework specifications; and (3) analyzed using comprehensive performance evaluation indices. The visual representation of this thesis' motivation, challenges and proposed solutions is illustrated in Figure 1.1.

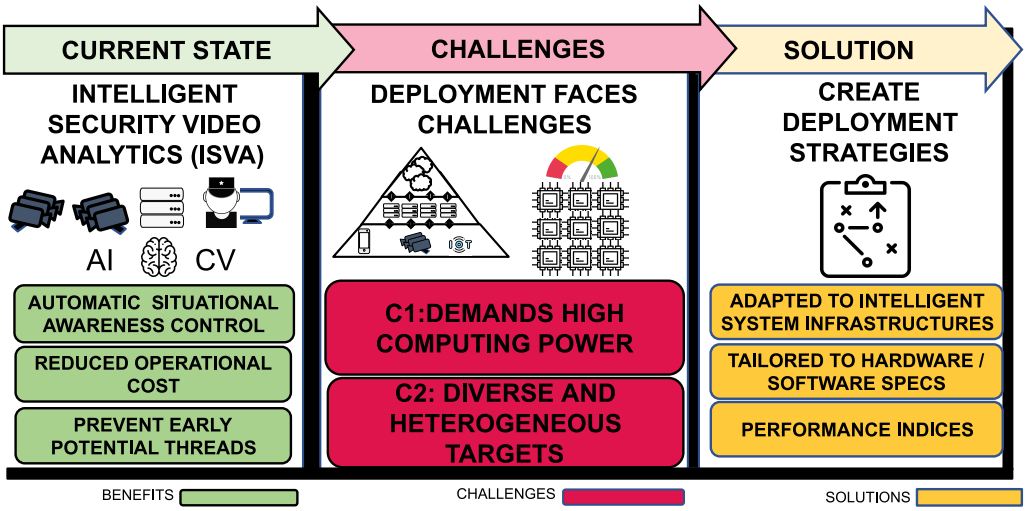


Figure 1.1: Visual representation of the Thesis' motivation, challenges and contributions.

Deploying DNN-based applications for ISVA system presents two main challenges. The first challenge is the high demand for computing resources required by these applications. The second challenge is how to deal with optimal deployment strategies with

diverse and highly heterogeneous ISVA targets. The following subsections analyze these challenges more in detail.

1.1.1 Challenge 1: DNN-based vision applications demand high computing resources

This challenge arises from three primary causes: DNN complexity, current hardware issues for DNN processing, and the lack of efficient parallelization techniques to execute multiple DNN instances.

As illustrated in Figure 1.2, DNNs are complex architectures composed of multiple layers of interconnected artificial neurons represented as a node graph. Adding prediction capabilities to DNNs requires a long learning process, which consists of an iterative training process updating the parameters of the neurons based on a large-scale image dataset. Once this learning process is finished, the DNN is prepared for prediction.

This prediction process, or *inference*, is the main bottleneck for DNN-based vision applications. It involves the propagation of artificial neurons to recognize highly complex patterns. These DNN architectures require storing a large number of parameters and operations which usually have 32-bit floating point (FP) precision. Consequently, processing the inference of all consecutive neurons in a DNN model hinders real-time processing, especially, for low-resource devices.

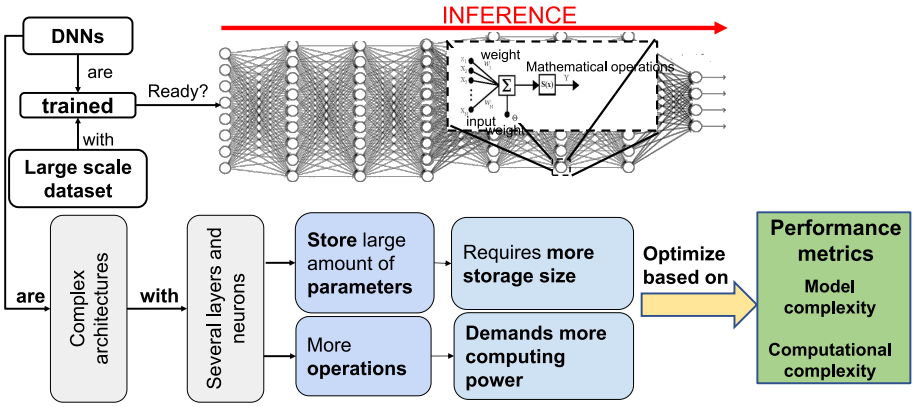


Figure 1.2: Key reasons to optimize DNN architectures.

To reduce the computing requirements of DNNs, numerous studies have explored optimization approaches. These methods assess the complexity of the DNN graph,

measuring model complexity (i.e., the total number of parameters) and computational complexity (i.e., the number of FLOPs) [1]. Model compression approaches such as pruning, quantization, and low-rank factorization [10] have been particularly effective for Convolutional Neuronal Networks (CNNs), significantly reducing both the storage size and computing demands of DNNs.

The DNN research community continues to propose new architectures to enhance learning capabilities of DNNs, which typically involves increasing model complexity. For example, vision transformers (ViT) have demonstrated superior accuracy to CNNs [11]. However, their increased complexity requires further research to make them suitable for low-resource devices. For example, this survey [12] demonstrates that ViT models can reach up to 239 millions of parameters and 1.5 gigabytes of storage size.

Choosing the optimal hardware is crucial for the successful deployment of DNN-based vision applications. While CPU and GPU architectures have traditionally been used for every computing task, the CPU's general-purpose design is not well-suited for DNN-vision applications. In contrast, GPU architectures are more suitable for these applications, but they can be expensive and energy intensive.

The lack of efficient hardware architectures for DNN inference has driven the development of specialized AI accelerators, including Google's TPUs, Intel's VPUs, etc. (xPUs in general) as discussed in [13]. These accelerators are designed to be energy-efficient and can scale up performance as the workload increases, as shown in Figure 1.3.

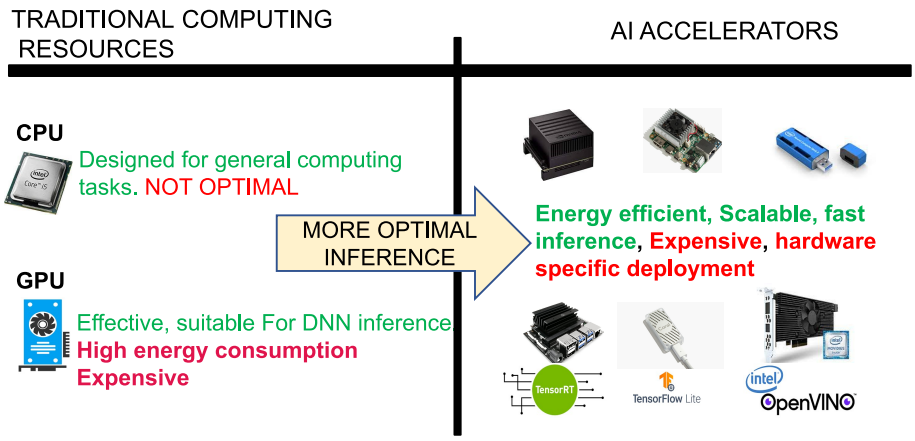


Figure 1.3: Traditional computing vs AI accelerators, benefits and drawbacks.

These accelerators are supported by deployment tools and inference engines, such

as TensorFlow Lite [14], TensorRT [15], and OpenVINO [16], which optimize the DNN models to fully utilize the hardware capabilities and accelerate the inference speed as noted in [17].

AI accelerators can be expensive, especially high-end ones, and, the benefits they provide may not be the same across all types of workloads. Moreover, deployment software tools and inference engines are typically vendor-specific and may pose challenges for users. Setting up and using AI accelerators effectively requires expertise in considering factors such as DNN model architectures, target hardware limitations, software / runtime limitations and performance requirements.

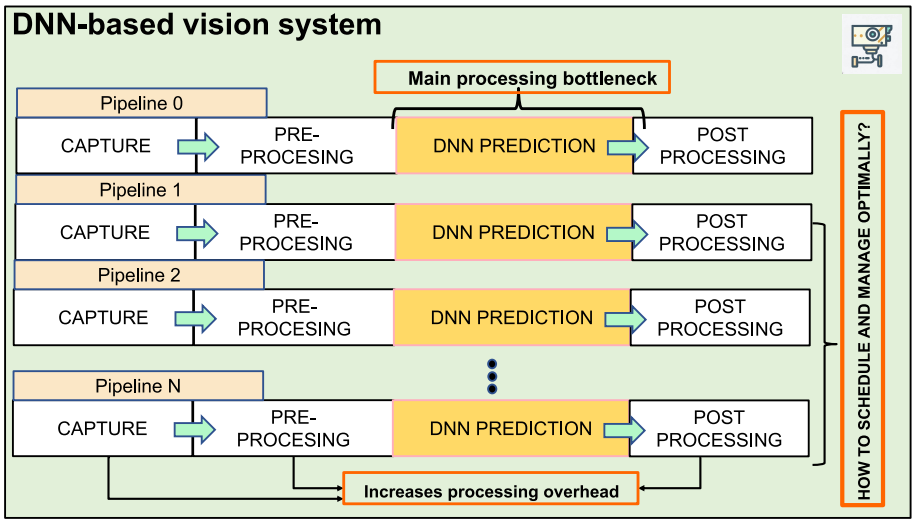


Figure 1.4: DNN-based vision pipelines scheduling issues, main bottleneck and processing overhead.

The DNN-based application workload is usually queued in a pipeline with image capturing, pre-processing, prediction tasks, and post-processing as shown in Figure 1.4. These extra image processing tasks significantly increase the processing overhead. Moreover, the integration of additional cognitive capabilities for video analytics involves executing multiple pipelines in parallel. Therefore, optimal handling and scheduling of these pipelines are crucial for maximizing hardware usage and making the acquisition of AI accelerators cost-effective.

1.1.2 Challenge 2: DNN deployment targets are diverse and heterogeneous

ISVA systems could consider a wide range of deployment targets, especially in the context of the Internet of Things (IoT), including distributed and decentralized systems, such as cloud, fog, and edge computing platforms [18]. As a result, there are theoretically numerous computer architectures, DNN models, and DNN deployment software tools that can be used.

However, in reality, many deployment targets face issues due to incompatibilities with hardware, DNN architectures, inference engines, and software tools. Therefore, deploying and maintaining DNN-based vision applications in such environments can be challenging, as illustrated in Figure 1.5.

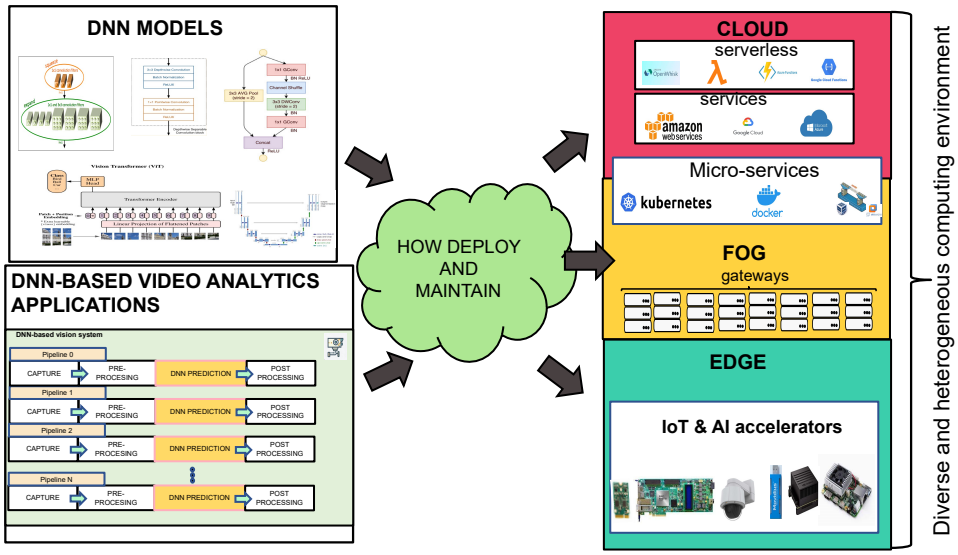


Figure 1.5: The complexity of deploying DNN-based vision apps in a diverse intelligent system infrastructure.

AI accelerators can be leveraged for edge computing to perform lightweight DNN processing, such as in smart AI cameras [19]. However, for many IoT devices, DNN inference processing is infeasible due to their limited computational capabilities.

To address these challenges, one solution is to delegate the inference processing to fog or cloud computing, where devices such as IoT gateways can handle a larger number of DNN inference requests [20]. This offloading strategy enables the execution of

heavy processing DNN-based applications in cloud environments with more resources.

The workload delegation process for DNN-based vision applications requires executing many tasks simultaneously. Until recently, task parallelization was commonly seen as a CPU multithreading monolithic application architecture. A monolithic application architecture refers to a software system that is built as a single, self-contained unit and executed in a single process. However, this architecture is not the most optimal choice for distributed environments. In contrast, microservices have emerged as a more suitable architectural pattern for modern infrastructures [21]. Microservices decouple a large application into a suite of small, modular services that can be developed, tested, and deployed independently. Thanks to microservices, DNN-based applications can be deployed as services [22], enabling easier scalability and finer-grained control over DNN inference requests. The differences between monolithic and microservice architectures are illustrated in Figure 1.6.

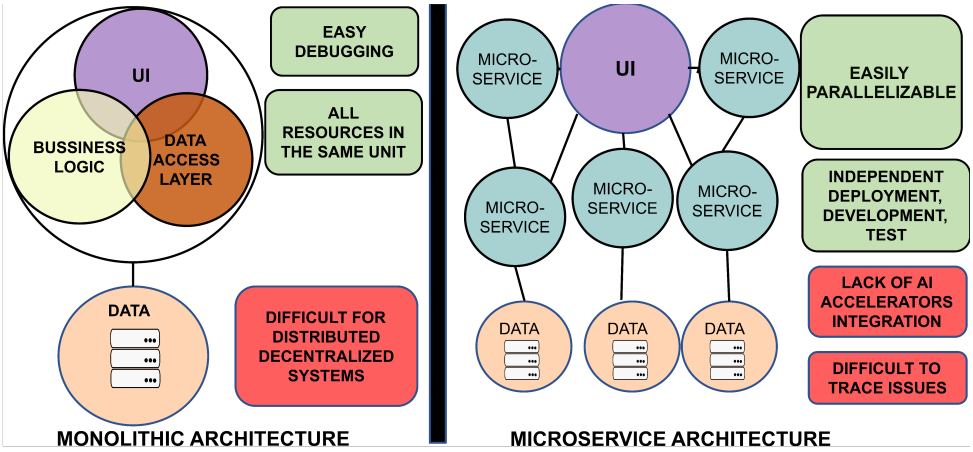


Figure 1.6: Comparing monolithic and microservice architecture: advantages and disadvantages.

A microservice architecture employs homogeneous runtimes for task execution, such as virtual machines, containers, and innovative cloud services such as serverless computing [23]. These runtimes provide benefits such as simplified maintenance and a homogeneous programming interface. However, they also have limitations, such as strict computing constraints and ephemeral storage resources, which can pose challenges for resource-intensive DNN-based applications. While CPUs and GPUs are typically used as the back-end hardware for these runtimes, newer AI accelerators are

not yet fully integrated into background computing. Additionally, allocating cloud GPUs for DNN processing inference can significantly increase the economic cost for the infrastructure.

1.2 Objectives

The main objective of this Thesis is to provide insights and contributions to the optimal deployment of DNN-based vision applications for ISVA systems. More specifically, this Thesis' contributions are focused on the generation and deployment of a modular processing pipeline in current intelligent system infrastructures. The modular design of this pipeline aims to leverage the strength of the customized deployments in order to improve performance indices. This dissertation is not only focused on the investigation of how to improve performance indices but also it aims to extend the key factors to improve the performance of ISVA applications such as subject, detection, recognition, and tracking.

More specifically, the objectives of this Thesis are as follows:

1. Develop a modular pipeline to optimally generate and deploy DNN models
    - Adapted to the needs of the intelligent system's infrastructures (i.e, centralized, decentralized, distributed with edge/cloud Computing).
    - Tailored to the software/hardware specs of targeted low-resource devices (RAM, CPU, storage size, operating system, etc)
    - Adjusted to the available DNN inference engines and deployment tools analyzing DNN structures and how to improve their performance indices.
2. Extract the key factors to enhance the efficiency of ISVA tasks such as object and subject detection, recognition and tracking.

1.3 Contributions

The contributions of this thesis are:

1. A new method for understanding the key factors to design DNNs with better performance indices considering the proposed DNN structures and the different kind of optimization techniques. This method was presented in Articulated Motion and Deformable Objects - 10th International Conference, AMDO 2018 [24].

2. A new method to evaluate the DNN models performance in serverless environments. More specifically, it provides a novel decomposition methodology from the current MLPerf benchmark to the serverless function execution model. This method was presented in the journal of IEEE software , in the special issue of Serverless Applications Engineering [25].

3. A novel approach for the optimal generation and deployment of DNN models for Cloud Serverless architectures for Intelligent Video Surveillance Systems. This approach was presented in the Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications (VISIGRAPP) [26].

4. A novel knowledge-driven approach for automatic deployment of DNNs for face recognition solutions in the highly heterogeneous IoT hardware. All of this, taking into account the specificities of user interaction, system security and privacy preserving issues. This contribution was reflected in three papers published in KES 2021 [27], Information [28] and IEEE Journal of Selected Topics in Signal Processing [29].

5. A comprehensive end-to-end approach for deploying a multi-DNN-based onboard video analytics system for edge computing environments. This deployment is based on a real-world pattern recognition scenario. In particular, this scenario involves checking the correct positioning of luggage in aircraft cabins in areas that could pose a safety risk during critical flight phases such as taxiing, take-off, and landing (TTL). This contribution outcomes are (1) a conference paper published in ROBOVIS2020 [30], (2) a patent application EP22382620, and, (3) a journal article submitted to IEEE Access (under rview process) and referenced in appendix section A.9, and (4) SMACS Dataset reference also in appendix section D.1.

1.4 Research environment and context

The research work reported in this Thesis was done during the participation of the PhD candidate in different research projects at the Intelligent Video Surveillance Department of Fundación Vicomtech, Basque Research and Technology Alliance (BRTA), under the supervision of Professor Ignacio-Arganda carreras (Computational Intelligence Group from the University of the Basque Country UPV/EHU) and PhD Luis Unzueta (Principal Researcher of Intelligent Video Surveillance Department of Fundación Vicomtech).

1.4.1 Projects

Related research projects of the thesis dissertation.

- SmaCS: The project developed a machine learning algorithm for cabin luggage control in poor light and contrast conditions. This project conceived a camera-based prototype solution, validated in a relevant environment in the CleanSky2 integrated cabin demonstrator, for digitalized on-demand verification of Taxi, Take-off and Landing (TTL) for cabin luggage.
  - Full-title: Smart Cabin System for cabin readiness
  - Funded under: H2020-EU.3.4. - SOCIETAL CHALLENGES - Smart, Green And Integrated Transport, H2020-EU.3.4.5.6. - ITD Systems.
  - Duration: 1 October 2019 - 31 July 2022.
  - Project coordinator: Otonomy Aviation (France).
- EWISA: The objective was to increase intelligence in video surveillance. The approach used was a process like entity, which consisted of camera-specific processes, a modular implementation of successive analysis layers for optimal situational awareness:
  - Full-title: Early Warning for Increased Situational Awareness
  - Funded under: FP7-SECURITY - Specific Programme "Cooperation": Security
  - Duration: 1 September 2014 - 30 June 2019.

  - Project coordinator: Kentro Meleton Asfaleias.

- SHAPES: This project aims to build an interoperable platform integrating smart digital solutions to collect and analyze older individuals' health, environmental and lifestyle information, identify their needs and provide personalized solutions that uphold the individuals' data protection and trust.

  - Full-title: Smart and Healthy Ageing through People Engaging in Supportive Systems
  - Funded under: H2020-EU.3.1. - SOCIETAL CHALLENGES - Health, demographic change and well-being, H2020-EU.3.1.4.1. - Active ageing, independent and assisted living, H2020-EU.2.1.1.3. - Future Internet: Software, hardware, Infrastructures, technologies and services.
  - Duration: 1 November 2019 - 31 October 2023.
  - Project coordinator: National University of Ireland Maynooth.

- NEOPOLIS: This project aims to create a surveillance system which analyzes the perception of citizen safety by tracking the number of people and classifying their gender using Deep Learning methods. In order to avoid a centralized server, this approach proposes to integrate each tracking system in the streetlights with a low-resource board to process each street locally and send the measurement through the net. Therefore, this project avoids privacy issues, due to the images are processed locally, and no image is stored.

  - Full-title: Solución inteligente de seguridad en base a interpretación semántica de percepción ciudadana y cálculo de flujos de información en tiempo real.
  - Funded under: Basque Government, Hazitek 2018-2019.
  - Duration: 1 september 2018 - 31 december 2019.
  - Project Coordinator: Iker Barrena (HISPAVISTA)

1.5 Thesis organization

This Thesis is structured as follows:

- Chapter 2 outlines the background and the related work presented in the state of the art.
- Chapter 3 exposes the optimal deployment of DNNs in serverless cloud architectures.
- Chapter 4 proposes a knowledge-driven methodology to deploy DNNs in the heterogenous IoT environment.
- Chapter 5 proposes a methodology to optimally deploy end-to-end computer vision tasks for multi-DNN video analytic pipeline, from problem definition, training, and the optimal pipeline generation of the video-analytic system.
- Chapter 6 defines the conclusions and the future work of this PhD thesis work.
- Part V defines this PhD thesis publications and patent applications.
- Part VI corresponds to the bibliography section that lists all the research references that were cited in the dissertation.

Part II

Related work

CHAPTER

2

Related work

This chapter provides a comprehensive analysis of the related work published in the literature that addresses the two challenges outlined in Section 1.1, namely: (1) the high computing requirements of DNN inference and (2) the complexity of heterogeneous deployment targets. In particular, we review the key contributions, gaps, and limitations in the field, highlighting the areas where further research is needed. This literature revision derives from [1], [2], [24] (contribution 1), [3], [31] articles and surveys.

The research community has proposed several techniques to address the first challenge, focusing mainly on optimizing three main aspects: (1) the DNN architecture complexity, (2) the AI hardware, and (3) the DNN inference and deployment software tools. More specifically, Section 2.1 analyzes different DNN complexity optimization techniques to reduce the DNN model inference processing time and storage size. Section 2.2 analyzes current AI acceleration hardware architectures along with their potential, characteristics, and limitations for deployment in intelligent system infrastructures. Section 2.3 reviews current DNN inference engines and deployment tools. In addition, regarding the second challenge, Section 2.4 describes different deployment approaches in heterogeneous infrastructures including cloud, fog, edge and IoT environments. Finally, the identified gaps and further research areas are discussed in Section 2.5.

2.1 DNN complexity optimizations

As highlighted in Section 1.1.1, optimizing the model and computation complexity of DNNs can help to reduce the DNN inference computing requirements [1]. This can be done by means of a compact network design, model compression (quantization and neuron pruning techniques), and tensor decomposition. Table 2.1 provides a summary of the most relevant complexity optimizations approaches for DNNs.

Table 2.1: Summary of current DNN complexity optimization methods.

Method	Type	Description	Layers	References
Compact network design	Comp. Efficiency	Network modifications	Conv, FC, Pool	[32], [33], [34], [35], [36]
Fixed point quantization	Compression	Bit-width reduction	All	[37], [38], [39],[40], [41], [42]
Codebook quantization	Compression	Weight sharing	All	[43]
Fine grained pruning	Compression	Unstructured sparsity	All	[44], [45], [43], [32], [46],[47]
Vector pruning	Compression	Structured sparsity	Conv kernels	[48], [49]
Filter pruning	Compression	Structured sparsity	Conv channels	[50], [51], [52], [53]
Low-rank factorization	Comp. Efficiency	Matrix decomposition	Conv	[54], [55], [56],[57]

2.1.1 Compact network design

Compact network designs focus on improving the architecture of current well-known DNN models [58] [59] to reduce their computational complexity, finding bottlenecks and providing a better architectural design. The DNN called AlexNet [46] was the winner of the ImageNet [60] Large Scale Visual Recognition Challenge (ILSVRC) 2012 with a remarkable difference in terms of accuracy with respect to its competitors. This work is considered one of the most influential papers in the DNN research field. Since its

publication, numerous new DNN architectures have been proposed by the research community in order to improve accuracy, but also trying to improve the number of parameters and operations. Figure 2.1 shows a benchmarking of the most representative DNN architectures proposed up to 2018 [1].

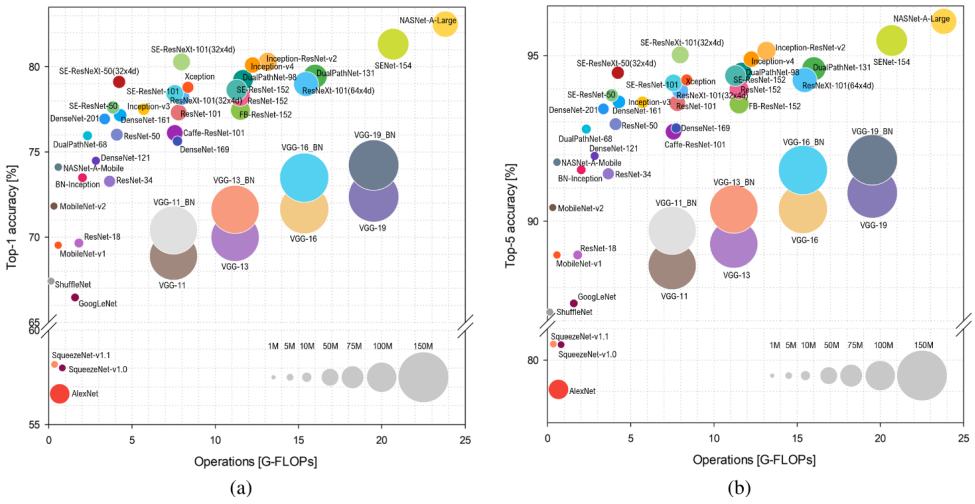


Figure 2.1: Benchmarking of the most representative DNN architectures published until 2018 [1].

SqueezeNet [32] is one of the first DNN models specifically designed with a reduction of the computation complexity in mind, while maintaining the accuracy. In particular, this DNN architecture defines a fire-module CNN architecture that came from AlexNet and had 50× times fewer MB. At the beginning of 2017, MobileNet [33] networks became popular in the field of object detection and classification, thanks to their depth-wise separable convolution strategy. This architectural improvement was integrated in large networks such as SSD [59] and VGG [58].

ShuffleNet [34] was later presented with the point-wise group convolution and channel shuffle approach. These new operations improved MobileNet's accuracy and reduced computational complexity to 40 MFLOPs.

Huang et al. proposed the DenseNet CNN architecture that utilizes dense connections between layers to alleviate the vanishing-gradient problem, strengthen feature propagation, encourage feature reuse, and substantially reduce the number of parameters [61].

In early 2018, Hasanpour et al. presented SimpNet [36], which featured a new pooling operation called SAF-pooling that improves the generalization power of the network, while it tries to keep the network structure simple by choosing the best features.

MobileNetV2 [35] was presented with a new architecture based on an inverted residual structure where the input and output of the residual block are thin bottleneck layers using lightweight depth-wise convolutions. MobileNetV3 [62] builds upon these V1 and V2 improvements and introduces new features such as the h-swish activation function and squeeze-and-excitation module. While these changes aim to enhance accuracy rather than speed, h-swish is faster than swish and helps enhance accuracy.

Google Brain researchers [63] proposed EfficientNet in 2019. The primary contribution of EfficientNet is that it achieves state-of-the-art performance on image classification tasks with significantly fewer parameters and computations. They proposed a new scaling method that uniformly scales all dimensions of depth/width/resolution using a simple yet highly effective compound coefficient. According to their experiments, the most accurate EfficientNet model achieves state-of-the-art 84.4% top-1 / 97.1% top-5 accuracy on ImageNet, while being 8.4x smaller and 6.1x faster on inference than the best existing CNNs. In 2021 this research group proposed EfficientNetV2 [64], a new family of convolutional networks that have faster training speed and better parameter efficiency than previous models, achieving 87.3 % of Top1 accuracy.

Recently, ViTs have outperformed state-of-the-art CNNs in terms of accuracy [12]. However, these models are complex and require significant processing power and storage space, making them less suitable for low-resource devices. Additionally, Han et al. [11] suggest that further improvements are needed for ViTs to be fully optimized for computer vision tasks, including better generalization and robustness, as well as a clearer understanding of why transformers perform well on visual tasks.

2.1.2 **Quantization**

Network quantization is a technique that compresses the original network by reducing the number of bits needed to represent each weight. This results in minimal accuracy loss during the prediction step. There are several methods of quantization, including fixed-point, binary, and codebook quantization [10].

Fixed-point quantization transforms 32-bit weights and activations to reduced bit-width values. In 2015, Gupta et al. [37] proposed Stochastic Rounding, demonstrating that a network could be trained with 16 bit-width weight values. In order to reduce bit-width, DoReFa-Net [38] was designed to train DNNs with a 6-bit gradient descent method applying a different adaptive quantization level depending on the layer, from 1-bit to 6-bit.

Reducing the quantization level to the minimum (1 bit per weight) can cause a significant decrease in the accuracy of inference prediction results. However, research such as BinaryConnect [39] has proposed binary training with values of 1 and -1 using Expectation Back-Propagation (EBP) to address these accuracy issues. To reduce computational complexity, several works such as Binary Neural Networks (BNN) [40] and XNOR-Net [41] have proposed using bit-wise binary operators instead of floating point operations (FLOPs). Local binary CNNs, as described by Juefei-Xu et al. [42], take a different approach to optimizing convolutions by using local binary convolution based on Local Binary Patterns (LBP) to process data.

Codebook quantization methods organize DNN weights using a codebook with groups of quantization codes. Each codebook value represents a quantization center, typically determined through k-means clustering. Rule-based strategies then apply predefined heuristics to set bit widths for convolution and fully connected layers. The model weights are shared and encoded using the Huffman method [43]. While the accuracy loss is negligible, these methods can increase DNN inference overhead due to the time required to search for encoded weights in the table.

2.1.3 **Pruning**

Network pruning aims at reducing network complexity to prevent over-fitting and obtain a better generalization. Based on the assumption that many parameters are redundant, or at least less relevant, this technique removes unimportant connections and therefore increases the sparsity of weights.

Pruning can be categorized into structured and unstructured methods. Unstructured pruning does not follow a specific geometry and requires additional information to define sparse locations. In contrast, structured pruning places non-zero parameters in well-defined locations without extra overhead.

The benefits and the drawbacks of the unstructured and structured pruning are:

- **Unstructured pruning:**
  - Benefits: Easy implementation, high sparsity and accuracy, higher pruning rate.
  - Drawbacks: Not hardware friendly, important performance degradation in highly parallel implementations like GPUs. This limitation can be overcome in structured weight pruning.
- **Structured pruning**
  - Benefits: Exploits hardware-efficient structures. It removes large sections of weights by changing tensor shapes such as channels and convolutional filters.
  - Drawbacks: Suffers from accuracy drop when the pruning rate is high.

Pruning methods can be classified based on their granularity level. Fine-grained pruning is used for unstructured pruning, while vector, kernel, and filter level pruning are used for structured pruning. Figure 2.2 provides a visual representation of each method according to its granularity level.

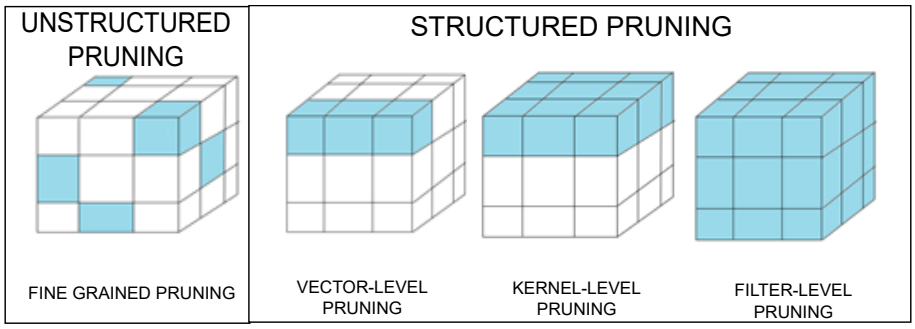


Figure 2.2: Visual representation of pruning methods. From left to right: fine-grained, vector, kernel and filter pruning.

Fine-grained pruning was explored even before the current DNNs by LeCun et al. [44] and Hassibi and Stork [45]. They used second order derivatives of the loss function, calculated the saliency, and removed parameters with low saliency. Following with

the unstructured fine-grained pruning, the lottery ticket hypotesis [65] conjectures that certain connections together with their randomly initialized weights can enable a comparable accuracy with the original network when trained in isolation. The previously mentioned deep compression approach [43] (section 2.1.2) combines fine-grained pruning, quantization and weight sharing through Huffman coding. This approach achieved 35× less MegaBytes (MB) and reduced 3× the FLOPs. In 2016, the SqueezeNet architecture [32] improved the deep compression approach by reducing AlexNet [46] by 50× less MB. Later, Abadi et al. [47] demonstrated that large-sparse models perform better for pruning optimizations than small-dense models across a diverse set of neural network architectures, improving the deep compression approach.

In vector level pruning, the current research (see for instance Mao et al. [48]) applies a better regularization of the sparse computation patterns, achieving similar accuracy to fine-grained pruning. For kernel level pruning, Anwar and Hwang [66] presented a structured sparsity approach. Each network connection importance is defined by the particle filtering approach and the importance weight of each particle is defined by the computed classification error of the network connectivity pattern.

Several works have focused on filter level pruning to improve the efficiency of ConvNets. For example, Li et al. [50] proposed a structured pruning method for all layers, reducing the FLOPs in ResNets by 20% [51]. Luo and Wu [52] introduced ThiNet, which prunes parameters based on the computed statistic information of the next layer. Ayinde et al. [53] suggested filter level pruning according to the relative cosine distances in the feature space. He et al. [67] proposed Progressive Soft Filter pruning, which improves accuracy by 1.08% and reduces the FLOPs of ResNet101 by 42%. This approach reduces the dependency on pretrained models and provides better performance when training from scratch.

Nevertheless, the benefits of network pruning are still unclear. The decision to choose the important weights is a very active topic and as a result, the rethinking of the value of the network pruning [68] opens a discussion about the assumptions made by the previous research. Liu et al. [68] make some observations that contradict common beliefs. For example, training a large and over-parameterized model is often not necessary to obtain an efficient model and learning important weights are typically not useful for the small pruned model. Also, the pruned architecture itself, rather than a set of inherited “important” weights, is more crucial to the efficiency in the final model, which

suggests that in some cases pruning can be useful as an architecture search paradigm. Finally inheriting weights from a large model is not necessarily optimal, and might trap the pruned model into a bad local minimum.

2.1.4 Low-rank factorization

Low-Rank factorization optimizes convolutional operations with matrix factorization methods. Lin et al. [54] presented a binarized CNN with separable filters where the convolutions were factorized by Singular Value Decomposition (SVD).

Tensor ring decomposition is an alternative to SVD, proposed by Qibin et al. [55], which represents a large dimensional tensor by circular multi-linear products through a low dimensional core sequence. Following this approach (Wide Compression: Tensor Ring Nets) Wang et al. [56] rely on tensor ring factorization for deep learning structures such as fully connected layers and convolutional layers. With the idea of a neural network automatic compression, DeepThin self compression [57] identifies and breaks artificial constraints imposed by low-rank factorization with a reshaping process that adds non-linearities on the approximation function.

2.2 AI acceleration hardware

The AI acceleration hardware's objective is to address the increasing demand for processing power for DNN inference required by AI applications. These hardware architectures are designed specifically to execute DNN inference workloads optimally. This results in faster processing times, reduced energy consumption, and improved overall performance. Based on the semiconductor market analysis published by [2], the AI hardware market would increase by five times in 2025 as depicted in Figure 2.3.

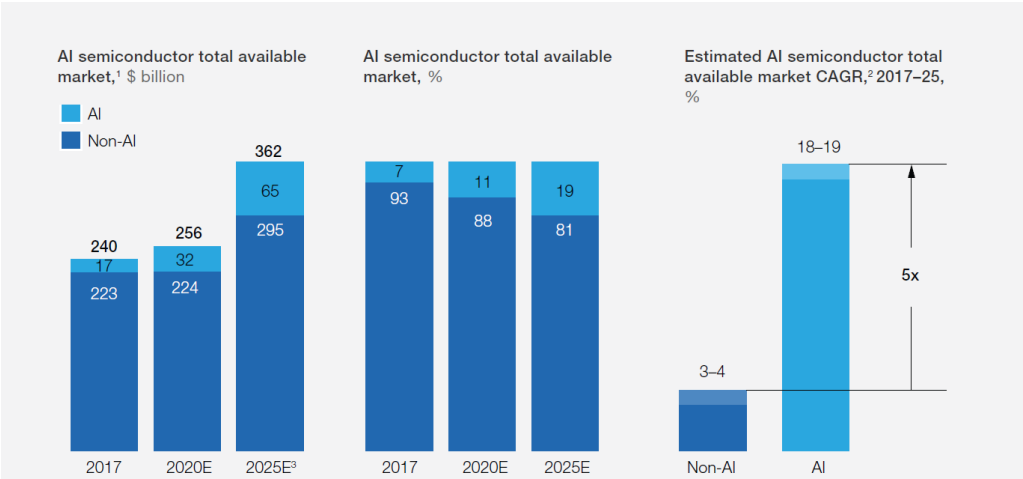


Figure 2.3: The expected growth of the AI-related semiconductor market. Source: [2].

The AI accelerator hardware typically involves specialized processing units (e.g., Google's TPUs [1], and Intel's VPUs [2], in general, xPUs) designed to perform specific AI computations and DNN layer operations, such as matrix multiplication, convolution and pooling. This heterogeneous hardware can be categorized into three main types [13]: Graphics Processing Units (GPUs), Field-Programmable Gate Arrays (FPGAs) and Application-Specific Integrated Circuits (ASICs). Below, a detailed description of each type of device is provided, including their unique benefits and drawbacks, in order to better understand their characteristics and suitability for different use cases and applications.

- **Graphics Processing Units (GPUs)**: Originally designed for graphics processing, GPUs are widely used for AI acceleration due to their highly parallel processing capabilities. These devices are very efficient for the convolution operations. However high-end GPU hardware can be expensive and require high energy consumption. The most representative example of GPU architecture for DNN inference is the NVidia Jetson family [3] because of the trade-off between computing capabilities and energy consumption.

[1] https://cloud.google.com/tpu/docs/tpus
[2] https://www.intel.com/content/www/us/en/products/details/processors/movidius-vpu/movidius-myriad-x/docs.html
[3] https://developer.nvidia.com/embedded/jetson-modules

- **Field Programmable Gate Arrays (FPGAs)**: FPGAs are programmable chips that can be customized for specific tasks and can be reprogrammed to adapt to different AI workloads, providing high performance and flexibility. However, the circuit reprogramming capabilities require a high level of expertise due to lower-level programming languages, which may not be suitable for all AI workloads. This is because the FPGAs have limited memory bandwidth and on-chip memory resources, which can result in performance bottlenecks for large DNN models. Additionally, FPGAs can be more expensive than GPUs due to their specialized hardware, reprogrammability capabilities, and development costs. Intel ARRIA 10 [4] is a representative example.

- **Application-Specific Integrated Circuits (ASICs)**: ASICs are highly specific integrated circuits designed to compute AI workload providing the best trade-off between performance and power efficiency. In contrast, designing and manufacturing ASICs can be expensive, especially for small-scale projects. ASICs main limitation is the lack of flexibility to adapt to AI workloads that require frequent DNN model and layer operation updates. The most representative ASIC hardware may be the Tensor Processing Unit (TPU) by Google and the Intel Movidius Vision Processing Unit (VPU) architecture.

The analysis of hardware performance for computing has been a significant and rapidly-evolving area of research in recent years. In particular, a comprehensive systematic literature review of AI hardware accelerators and ML tools was conducted by [69] , analyzing more than 169 research papers published between 2009 and 2019. This review has provided valuable insights into the recommendations and guidelines for AI hardware selection.

At the same time, both academic and industrial organizations have made several attempts at assessing the measurement of the inference performance of AI accelerators.. MLPerf [3] is an initiative by MLCommons to establish standard ML benchmarks for hardware, software, and cloud platforms. Even though this dissertation is focused on inference, MLPerf also does benchmarking for training tasks. MLPerf includes edge, mobile, datacenter and tiny hardware use cases for measuring inference performance

[4] https://www.intel.es/content/www/es/es/products/details/fpga/arria/10.html

based on their use cases and performance characteristics. From the first submission, MLPerf 0.5 [5], to the current MLPerf 2.1 [6], this benchmarking standard has been adopting different AI hardware devices.

Figure 2.4 illustrates the utilization and preference of various AI accelerators based on the MLPerf 0.5 submission [3] and the semiconductor manufacturing market analysis [2]. The results indicate that, despite the potential benefits of FPGA for AI workloads, its utilization remains low for several reasons. Firstly, high-end FPGAs can be expensive. Secondly, limited resources in terms of on-chip memory and arithmetic units, especially for CNNs, reduce their flexibility to use in AI workloads [70]. Lastly, reprogramming different AI circuits requires specialized expertise.

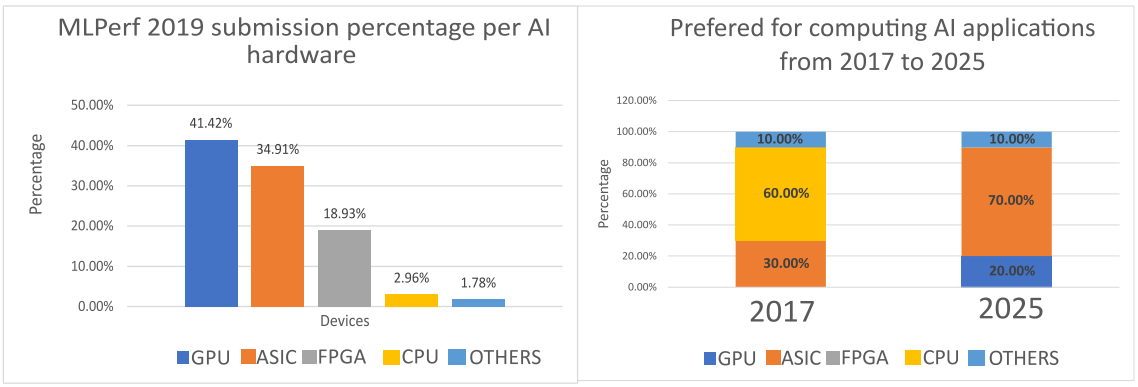


Figure 2.4: Overview of the future of semiconductors in MLPerf ML benchmarking standard and the current expectations of the markets on the preferences of use in the industry. Left: the submission percentage using AI hardware in MLPerf inference 0.5 benchmark [3]. Right: the preferences of the deployments for edge computing [2].

The preference for using GPUs and ASICs in AI applications can be attributed to the suitability of CNNs, which are widely used in computer vision. Convolutional layers are capable of learning complex hierarchical representations from input data. The highly parallel nature of GPUs and ASICs is well-suited to deploy AI workloads, as they are specialized for computing matrix operations [71]. For example, Google's TPUs can deliver up to 200 times the performance per Watt compared to traditional CPUs or GPUs. Intel's VPUs are optimized for tasks such as object detection and recognition and can deliver real-time performance at very low power consumption. However, compared to GPU

[5] https://github.com/mlcommons/inference_results_v0.5
[6] https://github.com/mlcommons/inference_results_v2.1

architectures, ASICs require lower weight precision (such as 8-bit or 16-bit), have limited flexibility, and can be challenging to adapt to new or changing DNN models and AI workloads, according to [37].

2.3 DNN inference optimization and deployment tools

Deploying an AI model with optimal performance requires the use of software tools that enhance inference speed. These tools make several optimizations to a DNN model that are crucial for achieving optimal inference latency. The optimization techniques are based on the DNN complexity reduction techniques described in Section 2.1 and they are usually optimized for ASICs, GPUs, CPUs, and FPGAs. There are currently many deployment tools, but most of these tools share the following characteristics:

- Graph optimization: Detects DNN common layer patterns to reduce the computation complexity such as tensor decomposition and matrix factorization techniques.
- Model quantization: Reduces the precision of the weights and activations in a model. They are usually reduced to 16 floating point (FP) and 8 int precision. The latter can lead to a significant drop in accuracy result. To mitigate this issue, many deployment tools include a calibration step which consists of a calibration of the weights and activations from a defined data subset.
- Graph partitioning: Splits the layers of a DNN into different parts that can be optimized for different AI accelerator. This process analyzes the affinity of each part and delegates to the most suitable AI accelerator. This method is especially effective for large DNN models.
- Parallelization methods: Mainly used for accelerating the DNN inference based on the hardware architecture design, operating system and DNN model.
- Dataflow optimizations: Reduce the latency time to access the data in memory. Improving the dataflow within a model, it is possible to minimize the time required to move data between layers and make more efficient use of the available memory.

There are several options for running DNN inference using deployment tools. Some tools, such as TensorRT [72], optimize the model for a specific accelerator. Others, like OpenVINO [73], use a plugin-based inference engine to support multiple NPUs. Finally, approaches like TensorFlow-Lite [14] generate both the model and the inference engine for each deployment, reducing the overall space requirement by only including necessary operations. There are tools available for general-purpose inference, such as open neural network exchange (ONNX) runtime [74]. These include various optimized inference engines that are tailored to specific target hardware. ONNX defines different runners that can be configured with different drivers and hardware, such as NVidia GPUs and Intel CPUs.

Table 2.2 compares the common features of TensorRT, OpenVINO and TensorFlow-Lite, ONNX runtime DNN deployment tools.

Table 2.2: Comparison of optimizations in the TensorRT, OpenVINO, and TensorFlow-lite DNN deployment tools.

Optimizations	**TensorRT**	**OpenVINO**	**TensorFlow lite**	**ONNX runtime**
Inference	HW specific model	Plugin based execution	Target code generation (XLA)	ORT optimized , ONNX for general
Graph optimizations	No	Fused operators	Fused operators	fused operators
Model quantization	16FP/8Int	16FP/8Int	16FP/8Int	16FP/8Int data-layouts
Layer decomposition	DLA with GPU fallback	Heterogeneous plugin	Model Tiling	NO
Parallelization methods	Multi-stream execution	Asynchronous inference	Depends on HW	Operation threading
Dataflow optimizations	Dynamic Tensor Memory Kernel auto-tuning Layer and Tensor Fusion	NO	NO	Shared memory General purpose allocator
Target hardware	GPU, ASIC FPGA, VPU	CPU, GPU, TPU	Mobile (CPU, GPU)	Depends on general runner CPU & GPU

There have been studies on the evaluation of deep learning deployment tools and inference engines. For instance, Ulker et al. [4] evaluated the inference performance of tools such as TensorRT, OpenVINO, and TensorFlow-Lite using state-of-the-art CNN architectures for multiple hardware platforms.

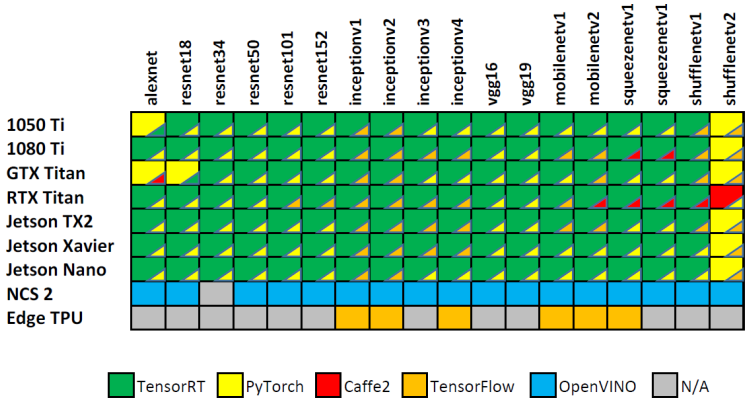


Figure 2.5: Software tools delivering minimum execution time for each platform-network configuration. Source: [4].

This paper [4] reveals important insights and conclusions for optimal tool selection when considering minimum latency and maximum throughput. As can be seen in the results shown in Figure 2.5 TensorRT has the best overall performance for inference on NVidia platforms. According to their insights, PyTorch performs better in small batch sizes and compact networks while TensorFlow delivers higher throughput in higher batch sizes. Finally, they conclude that none of the software is always the best choice. Shafi et al. [75] analyzed the performance between TensorRT software optimizations on the NVidia platform GPUs hardware. Through an empirical study, based on advance driver assistance system (ADAS), they observed significant performance and accuracy gains from software optimizations, with some additional highly unexpected non-deterministic behaviors, such as different outputs with the same inputs or increased execution latency for the same neural network model on more powerful hardware platforms.

Li et al. [76] surveyed deep learning compilers and their design, including multi-level intermediate representations (IR) and background/foreground optimizations. They presented a detailed analysis and highlights potential and research directions of deep

learning compilers using TensorFlow-Lite and TensorFlow-TensorRT conversions. They found that TensorFlow to TensorRT (TF-TRT) direct optimization performs better at high precision, especially with GPUs with tensor core ASICs. However, TensorFlow-Lite performs better for lightweight DNN models.

Gorbachev et al. [73] and Demidovskij et al. [77] presented the OpenVINO DL Workbench, the platform that provides the performance tuning, analysis, and deployment tools for the deployment of DNNs on Intel hardware for several NPUs such as CPUs, GPUs, VPUs, FPGAs. According to the authors, the DL Workbench tends to play one of leading roles in terms of feature completeness across other existing platforms in the field.

2.4 Deployment heterogeneity

An intelligent system infrastructure consists of both hardware and software components that are capable of performing complex tasks requiring human-like intelligence. To meet the high processing demands of these tasks, a multi-level computing environment may be utilized. This computing structure, as shown in Figure 2.6, can be divided into three levels: cloud, fog, and edge computing.

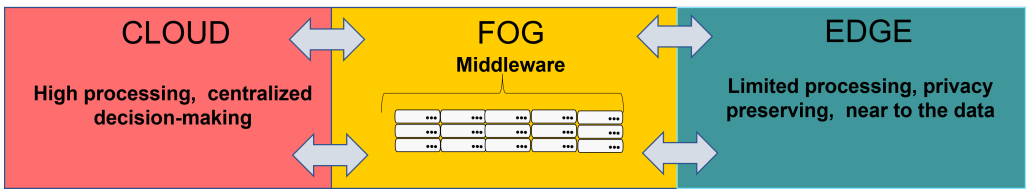


Figure 2.6: The computing environments for the intelligent system infrastructure. Cloud for centralized and high processing, Fog environment for middleware tasks, Edge for processing the information near the data.

Cloud computing is responsible for taking centralized decisions, while edge computing processes data close to the sources to avoid network latencies and maintain privacy. The fog computing layer plays a crucial role in bridging the gap between cloud and edge computing working as a middleware. The mission of fog computing is to reduce the workload overhead of cloud computing while also supporting the AI services that edge computing cannot handle.

This cloud, fog, edge multi-level computing structure is an ideal target for deploying end-to-end DNN-based vision applications across different industries, including ISVA systems. But, as previously discussed in Section 1.1.2, the growing heterogeneity and diversity of intelligent system for deployment infrastructure present a challenge. This heterogeneity can be attributed to several factors:

- Hardware architectures with heterogeneous computing capabilities, ranging from specialized micro-AI hardware accelerators to powerful cloud computing servers.
- The need to scale AI services to meet the demands of expanding user bases further contributes to the increasing number of deployment targets.
- Advances in DNN inference architectures, engines, and deployment tools continue to evolve, leading to a constant expansion of deployment targets.

The deployment of DNN-based vision applications in ISVA systems is an area that is still in its early stages, as noted in a systematic review by Shidik et al. [6]. This work provides an in-depth explanation of the current datasets, integration, and network infrastructure design required to meet the demand for multimedia data. According to Limna et al. [78], there are several proposals to move traditional video surveillance to intelligent video surveillance as a service (VSaaS) which would represent the future of ISVA systems. But, the current approaches are merely based on the recording and video analysis using commercial applications. This Limna work proposes a component-based VSaaS called Nokkhum which deploys complete video surveillance through REST APIs, webfront client, controllers, data communication, and image processor. Although this image processor includes face detection and motion detection, it is primarily focused on inter-module communication and information sharing, rather than visual analytics deployment.

Despite the lack of specific articles of DNN deployment in ISVA systems, the researchers instead propose approaches for general-purpose deployment of DNN-based vision applications in cloud, fog, edge, and IoT environments. The following subsections describe the current literature on each computing environment.

2.4.1 Cloud based deployment

Serverless computing is seen as the next step in the evolution of the cloud computing model [31]. Serverless computing enables dynamic management of resource scalability under the Function-as-a-Service (FaaS) paradigm. Developers can delegate operational complexity and scalability to the cloud provider without requiring extensive cloud computing expertise.

Despite its limitations, the recent literature has explored the deployment of DNNs in serverless environments providing important insights and conclusions [31, 79, 80].

Ishakian et al. [31] evaluated the feasibility of deploying a ML serving system based on the DNN inference for image classification tasks. They concluded that the performance of these models is acceptable when the serverless container is constantly working, but the when the container is in the initialization process, the performance is not under service level agreements (SLA). They also argue that serverless computing limitations such as the lack of GPUs, the stateless nature of the functions and the difficulties in maintaining the state between invocations make the DNN deployment very challenging.

Zhang et al. [79] presented MARK, a general-purpose and cost-effective inference serving ML system built in a general cloud environment. This work employs three design choices to tackle the challenge of meeting service-level objectives (SLOs). Firstly, it dynamically batches the inference. Secondly, it applies predictive auto-scaling to hide provisioning latency. Finally, it provides flexible structure to cover occasional workload spikes using GPU hardware and serverless function instances.

Romero et al. proposed the INFaaS [80] inference serving system. This system automatically generates and uses model-variants, which are versions of a model that vary in terms of resource footprint, latency, cost, and accuracy. INFaaS also selects the appropriate hardware architecture and makes scaling and resource allocation decisions to meet user-defined objectives.

The automatic scalability of serverless computing offers a promising solution for running DNN-based vision applications within a micro-service architecture. However, these applications typically operate within a homogeneous runtime environment and there is still a tendency to use monolithic applications for cloud computing. Additionally, workload prediction remains an ongoing challenge in this field.

2.4.2 Fog and edge deployment

In contrast to cloud computing approaches, edge and fog DNN deployment alternatives are focused on harnessing the heterogeneous capabilities of AI hardware. These strategies primarily propose DNN inference scheduling methodologies within monolithic application architectures. As stated in the Section 1.1.2, a monolithic architecture refers to a self-contained program in which all resources such as the main program, libraries, and files reside within the same execution unit.

The popularity of these approaches for video analytic pipelines has promoted the release of commercial applications such as NVidia DeepStream [81] and Intel DL Streamer [82]. Both are based on the open-source GStreamer [7] multimedia framework, and they provide hardware-accelerated modules that encompass decode, preprocessing, and DNN inference of input video streams. They consider the pipeline for object detection, attribute classification, and tracking but also allow users to build more complex pipelines if required. However, deploying multiple concurrent classification DNNs as in the case of ISVA systems is challenging in current off-the-shelf DNN accelerators and deep learning frameworks, as they are not designed for that. They only provide a single-level priority, one-DNN-per-process execution model, sequential inference interfaces, and assume that the DNN inference is executed on a single processing element, CPU, GPU, or xPU; no more than one simultaneously.

Some works have focused on scheduling the heterogeneous computing capabilities for embedded processors. For instance, Xiang et al. [83] proposed DART, a CPU-GPU scheduling framework for DNNs offering deterministic response time to real-time tasks and increased throughput to best-effort tasks. It employs a pipeline-based scheduling architecture with data parallelism, where heterogeneous CPUs and GPUs are arranged into nodes with different parallelism levels. Similarly, Lim et al. proposed ODMDEF [84], an on-device CPU-GPU co-scheduling framework to remove the performance barrier precluding DNN executions from being bound by the GPU. Experiments with the NVidia Jetson AGX Xavier platform show that it speeds up the execution time by up to 46.6% over the GPU-only solution. Jeong et al. [85] proposed a parallelization methodology to maximize the throughput of a single DNN-based application using GPU and NPU by exploiting various types of parallelism on TensorRT: multi-threading, multi-stream,

[7] https://gstreamer.freedesktop.org/

pipelining of the inference network, and partial network duplication. They devised a heuristic to determine the pipeline cut-points achieving 81%-391% throughput improvement over the baseline inference that uses the GPU only in six real-life object detection networks on a NVidia Jetson AGX Xavier board. These same authors also presented a tool called Jetson-aware Embedded Deep learning Inference acceleration (JEDI) for making the proposed optimizations in NVidia Jetson boards [86].

There are alternatives focused on optimizing the deployment of multiple DNNs in a single processing element. For example, Kim [87] presented a methodology to allow higher priority DNNs to occupy the GPU preferentially. Every decomposed DNN layer job is stacked in a priority order inside the layer queue. Higher priority layer jobs can preempt lower-priority ones using stream prioritization, reducing the execution time of DNNs by up to 60.4%. Cox et al. proposed MASA [88], a responsive memory-aware multi-DNN execution framework on CPU, an on-device middleware featuring modeling inter and intra-network dependency, and leveraging complementary memory usage of each layer. It can consistently ensure the average response time when deterministically and stochastically executing multiple DNNs. Finally, Yu et al. [89] proposed a graph- and runtime-level cross-layer scheduling framework for multi-tenant inference optimization in GPU, which automatically coordinates concurrent DNN computing at different execution levels. It achieves 1.3x 1.7x speedup compared to regular DNN runtime libraries (e.g., CuDNN, TVM) and concurrent scheduling methods (e.g., NVidia Multi-Stream).

2.4.3 **Deployment for IoT environments**

IoT paradigm refers to the network of physical objects connected to the internet and other devices or applications. These objects are embedded with sensors and other technologies that allow them to gather, send, and exchange data, making them smarter and more interactive. While the previous subsection analyzed the optimal deployment of DNN-based vision applications per computing layer (cloud, edge, fog), there are other research works focused on the optimal deployment of DNN-based vision applications across multiple interconnected devices in an IoT environment.

The benefits of the AI integration in IoT technologies had been recently reviewed by Chang et al. [90] who conducted a comprehensive analysis of the potential and chal-

lenges of a new paradigm called artificial intelligence internet of things (AIoT). The authors explored the benefits of AIoT, including enhanced efficiency and productivity, improved quality of life, and reduced environmental impact. They also identified several challenges, including data security, privacy concerns, and lack of standardization.

The recent published literature presents an increasing interest to deploy face recognition in IoT systems. The majority of the works propose secure and efficient user authentication approaches based on computer vision and AI algorithms for these environments. The comprehensive review of Yousefpour et al. [91] is usually referenced as an article that covers secure user authentication along with many other IoT-related paradigms such as fog computing, cloudlets, and multi-access edge computing (MEC).

All these approaches handle the security and the performance of the data produced by the IoT platform. Hu et al. [92] presented an example of a fog-based face identification and resolution system. Instead of using DNNs for images processing, this work proposed classic Haar and LBP features for face detection and identification. Although these methods use less computational resources, these computer vision methods are not as accurate as DNN face recognition techniques. To reduce the network traffic, this framework delegates part of the resolution task to fog nodes, and only the biometric data is transmitted to the cloud. Using a task partitioning strategy, the cloud overhead is relieved and the devices located at the edge network assume the role of image processing, making full use of the computing power.

In [93], Hu et al. extended this approach to solve confidentiality, integrity, and availability issues. The proposed method provides a mechanism for an authentication and session key agreement scheme, supported by a data encryption scheme and data integrity checking. Wang and Nakachi [94] presented a secured framework for face recognition in edge and cloud networks based on sparse representation. In contrast to DNN based face recognition, this method is based on a discriminative dictionary, which requires fewer computation resources, but the accuracy results are worse than DNN based face-recognition approaches. Another inconvenience of this method is that each time a new sample is added, dictionary training is required. these authors establish a distributed learning framework to train the recognition method. The training samples are divided into two parts, one for dictionary and classifier training, and one for ensemble training. The decision template is extracted for each class in the intermediate space, expanded by the estimated label vectors and based on the ensemble training set. The

recognition is identified according to pairwise similarity between the decision profile of the testing sample and each of the decision templates. To guarantee privacy, they adopted a low complexity for the encrypting algorithm, based on random unitary transform, without affecting the accuracy. Mao et al. [95] designed an edge device-based DNN training scheme for face recognition with differentially private mechanisms to protect private data. The DNN is split into two parts, one deployed on the user's device and the other on the edge server. They avoided cryptographic tools to keep the user side lightweight.

2.5 Discussion

Our analysis revealed the following outcomes and gaps for the optimal deployment of DNN-based vision systems from various perspectives:

- Although there are ML benchmarking standards for measuring single DNN inference performance in several target hardware such as MLPerf [3], **measuring ISVA system performance requires more comprehensive evaluation than analyzing the inference latency, accuracy and energy consumption**. ISVA systems often involve the simultaneous execution of multiple DNNs, and thus, their behavior under such conditions needs to be assessed. This includes evaluating the performance of executing several models simultaneously and how they interact with each other. Chapter 3 defines a methodology to make the comprehensive assessment of the DNN inference based on serverless potentials and limitations.

- **In the intelligent system infrastructures, the DNN inference scalability issues are typically addressed using monolithic architecture relying on task scheduling and workload prediction**. However, there is room for improvement in leveraging the automatic scalability capabilities and dealing with the limitations of the micro-service application architecture. Chapter 3 proposes different methodologies to deal with this limitation for DNN-based vision applications for ISVA systems.

- The general-purpose DNN deployment approaches **are primarily centered on optimizing single DNN model for specific hardware or runtime**. Also, the deployment tools and inference engines require high expertise to take the full

advantage of the optimizations, especially for ASICS. Chapter 4 proposes an automatic deployment algorithm based on knowledge-driven methodologies to automatize this heterogeneous deployment process.

- Most of the **DNN-based vision optimizations are focused on DNN architecture complexity reduction**. However, there is a lack of attention given to other important tasks, such as camera capturing and image pre- and post-processing optimizations, which are essential for the end-to-end DNN-based vision applications. Chapter 5 proposes a comprehensive end-to-end methodology for deploying a DNN-based vision application into a modular and auto-configurable video analytic pipeline.

- While there are commercial alternatives for **general-purpose video analytics pipelines, they mostly focus on computer vision tasks such as object detection, tracking, and classification**. There is a wider range of DNN-based end-to-end applications in ISVA that require research attention. This limitation is also analyzed in Chapter 5.

Part III

Research Results

CHAPTER

3

Optimal deployment of DNNs in serverless cloud architectures

This chapter explores the optimal deployment of DNNs in serverless cloud architectures. As analyzed in Section 2.4, serverless computing is seen as the next step in the evolution of the cloud computing model, and its benefits can be crucial to enhance the scalability of AI services. This chapter refers to contributions 2 and 3, described in Section 1.3. More specifically, this chapter is divided into three different sections:

- The background and challenges for the deployment of DNN-based vision applications in serverless environments (Section 3.1).
- An approach to benchmark DNN-based applications performance in serverless cloud architecture based on the MLPerf standard rules [3] (Section 3.2).
- An approach for the optimal deployment of multiple DNN models within a serverless environment for use in a Video Surveillance System (VSS). (Section 3.3).

3.1 Background and challenges

Serverless computing is a cloud-native model under the scope of the Function-as-a-Service (FaaS) paradigm. This computing model hides server usage from developers

and runs code on-demand, automatically scaled and billed only for the time the code is running [31]. Its two key features are cost (pay-as-you-go billing with millisecond granularity) and elasticity (scaling from zero to "infinity"). It allows developers to concentrate on providing a piece of code (function) to be executed by the serverless computing platform and to delegate all their operational complexity and scalability to the cloud provider, without requiring a high level of cloud computing expertise.

The FaaS platforms are materialized in function instances. More specifically, when a function is called, the serverless computing platform creates an instance of that function to execute the code. These instances are created on-demand and are typically short-lived.

FaaS platforms have two stages [96]. The first stage begins when the FaaS function is invoked for the first time, creating an isolated runtime environment with the necessary resources. This process takes additional time to be completed and, consequently, this stage is called cold start stage. When the container initialization is finished, the remaining function instances are executed concurrently. This second stage is called warm stage. Serverless functions offer several advantages that make them well-suited for DNN prediction tasks. However, deploying DNN models in a serverless environment presents several challenges:

- **Serverless functions are stateless by default**. This means that they do not store any information about previous executions or interactions. Therefore, in contrast to monolithic applications, debugging and tracing serverless functions can be complex due to several reasons. Serverless functions are managed, deeply controlled, and disconnected environments. Developers do not have most of the typical debugging tools at their disposal. The distributed and controlled nature of the environment essentially blocks any chance of remote debugging. The increased complexity, loss of control over software layers, and the large number of participating functions and back-end services complicate the task of finding the cause of a faulty execution. A tedious but widespread strategy is the manual analysis of log data. Controlling the state requires the use of an external service which provides that state. But adding a state to the application can increase the complexity of the solution, because it makes it harder to scale, and it can potentially create a single point of failure.

- **The lack of access to GPUs can be a significant limitation for serverless computing**, especially for workloads that require intensive parallel computing such as DNN-based vision applications for ISVA. Without access to GPUs, serverless computing platforms are limited to using CPUs backend hardware [97], which can significantly slow down the execution of parallelizable tasks. This limitation can result in longer processing times, higher latency, and increased costs for users who need to scale their workloads quickly. There are very recent initiatives for enabling GPU in serverless environments [98], but they are still far from a robust integration. As explained in Section 2.2 GPUs are specialized hardware that can perform parallel computations much faster than CPUs. Even when GPU-based serverless resources are available, they may be more expensive than CPU-based instances. Therefore, the deployment strategy should consider the most cost-effective computing platform for the specific use case.

- Each function is executed in an isolated container that is created specifically to handle a single request. As a result, **functions rely on ephemeral storage during the execution of the function**. Therefore, when the function instance finishes, all data in the container is erased. This means that if a function fails or is terminated for any reason, all the data is lost. This can be a problem for DNN inference, where data consistency is critical. For instance, if a function fails while processing an input, the entire inference process may need to be restarted due to the loss of output information. Additionally, serverless computing platforms often impose limits on the amount of ephemeral storage available to functions. This can cause issues for DNN inference where models and data can quickly exceed the available storage. If a function runs out of storage, it may fail or need to be restarted, resulting in increased latency and reduced performance.

- The serverless system creates a container and loads the function code into memory when a function is invoked for the first time. This initial setup time, also known as **cold start, can result in increased latency for the first request**. Cold start time can be a significant limitation for DNN inference particularly for real-time applications, making it unsuitable for time-critical applications. Additionally, DNN inference tasks may require a large number of compute resources, leading to longer cold start times. This increased latency issue is particularly relevant when

loading larger DNN models. The longer the cold start time, the more resources are consumed, which can lead to higher costs and reduced scalability. Another drawback of the cold start issue is the inability of serverless environments to effectively manage sudden increases in traffic. In traditional server environments, administrators can pre-scale infrastructure to accommodate high loads. However, the dynamic scaling nature of the serverless systems can lead to increased cold start.

- **When the serverless computing functions are in the warm state, they can scale rapidly to meet spikes in demand, but they cannot scale up to infinity**. However, there are limits to the resources that can be allocated. If demand exceeds these limits, the system may struggle to handle the load. Scaling up resources in serverless platforms requires allocating more resources to functions, which can increase costs. However, configuring larger resource functions do not necessarily result in better DNN inference performance. This is because the target hardware configuration is managed by a serverless cloud provider.

In order to deal with these limitations, we summarize the current performance strategies presented in the literature [96] [99] for serverless environments:

- Concise function logic: Splitting high complex and fat/monolithic serverless functions into smaller serverless functions can reduce the processing time and memory consumption. Also, separating each serverless function entry logic (handler function) from the core logic (core function) leverages cold start time reduction. Streaming services and online storage services can optimize data transformation and the request payload size.

- Third-party dependencies: If third-party dependencies are required, avoid using open source packages. Since their general-purpose and third-party interdependency nature, open source packages include more functionalities than required and, thus, can cause a significant slowdown in cold start time and increase processing time.

- Resource management: Limit the re-initialization of local variables in every instance. Instead, global/static variables or singleton patterns can be used to handle

the application scope variables. This approach can help reduce the startup time of serverless functions and improve their performance. Using global or static variables in serverless functions can introduce some complexity, as the variables need to be properly initialized and that their state is managed correctly across multiple invocations. However, with careful design and implementation, this approach can help improve the performance of serverless functions.

- Allocated function memory: The amount of memory allocated to a function can have a significant impact on serverless performance and execution cost. Allocating more memory to a function can improve its performance by reducing the time it takes to execute, but it can also increase the cost of running the function. Finding the optimal balance between the amount of memory allocated to a function and its execution cost can be challenging. It is important to carefully evaluate the memory requirements of the function and to monitor its performance to ensure that it is using the allocated memory efficiently.

- Language agnostic advice: The choice of programming language can have a significant impact on the performance and execution cost of serverless functions. Different languages have different characteristics, such as startup time, memory usage, and ease of deployment, that can affect the performance of serverless functions. Finding the optimal programming language for a given serverless application involves balancing the configuration of computing resources with the execution cost. A key strategy is to carefully evaluate the requirements of the application and select a language that is well-suited to the task at hand.

- Keep the container in the warm state: Make pre-configured periodical calls to serverless functions to avoid changing to a cold stage. Keeping the containers in a warm state, it's possible to reduce the latency associated with cold starts and improve the performance of serverless functions. This approach can be particularly useful for applications with low or unpredictable traffic patterns, where cold starts can have a significant impact on performance.

In the following section we propose a benchmarking methodology to evaluate the performance of DNN inference in a serverless environment. This evaluation process

is essential to define the optimal deployment strategies for DNN-based vision applications.

3.2 Benchmarking DNN inference in serverless environments with MLPerf

Measuring the performance of DNN-based vision applications across diverse hardware components in an intelligent system infrastructure is complex. Each hardware/runtime system has unique capabilities and performance characteristics, making accurate comparison difficult. Organizations such as the MLPerf consortium, a group of AI leaders from academia, research labs, and industry, aim to solve this problem by building fair and useful benchmarks for unbiased evaluations of training and inference performance. The MLPerf Inference benchmarking suite has become the standard, driven by over 30 organizations and 200 ML engineers and practitioners [3]. However, MLPerf benchmarking tools are currently designed for monolithic architecture scenarios. This means that the DNN inference is processed in the same execution unit (the targeted CPU, GPU or xPU), thus colliding with serverless system architectures. In this chapter, we propose a benchmarking methodology of DNN inference in FaaS platforms taking into consideration the MLPerf practical guidelines and rules. More specifically, this approach aims to answer the following questions:

- **How to deploy DNN models and their inference engine?** This includes choosing a suitable DNN inference engine and loading and processing the trained DNN models, considering the space and memory constraints of serverless platforms.

- **How to handle the performance results?** Considering that the FaaS platforms are stateless with ephemeral storage, how do we manage the persistence of the measured results?

- **How to design a benchmarking platform on ML applications in serverless environments?** Traditional ML benchmarking methodologies measure the performance of ML models in physical target hardware in monolithic application architectures. In contrast, the serverless environment has a distributed nature

designed for micro-service application architecture. Also, these serverless applications are deployed to virtualized and containerized environments. Therefore, these differences require rethinking the benchmarking platform design.

3.2.1 Enabling serverless runtime in MLPerf

MLPerf Inference is designed to benchmark common ML-based vision tasks (image classification and object detection), and NLP tasks (translation). MLPerf provides a comprehensive submission process to contribute to the benchmarking results for new hardware architectures. To do this submission, MLPerf Inference provides a benchmarking suite with software tools to enable fair comparisons between hardware targets and DNN models.

The MLPerf Inference architecture contains three components:

- System Under Test (SUT): It runs the DNN inference and the performance measurements are sent back to the Load Generator (LoadGen). The submitter is responsible for implementing the System Under Test (SUT), with rules in place to ensure model-equivalence across different architectures and frameworks. Techniques such as pruning and retraining are not allowed since benchmarking is based on pre-trained models. Additionally, caching images and manipulating or optimizing the image acquisition process are prohibited.

- LoadGen: This component is responsible for feeding input data to the System Under Test (SUT) and calculating performance measurements for benchmarking. Its behavior is determined by a configuration file that is read at the start of benchmarking. LoadGen generates query traffic based on MLPerf scenarios and collects information for logging, debugging, and post-processing. It records queries and responses from the SUT and reports statistics and summarizes results when finished.

- Dataset: To prepare input data for benchmarking, MLPerf uses standard and publicly available datasets to enable community participation. However, only a subset of the dataset is used for benchmarking. The LoadGen component is responsible for downloading these datasets from public sources.

MLPerf Inference (version 2.1) considers the following application scenarios for benchmarking when feeding the SUT:

- Single stream: In this scenario, an inference query is sent, and the next query is only sent after the first one is completed. This setup is used in applications where quick response times are critical, such as when multiple client applications require prompt results.
- Multi stream: In this scenario, LoadGen periodically sends a set of inferences per query at intervals between 50 and 100 ms. This covers a range of inferences for various industrial automation and remote sensing tasks. For example, video analytic pipelines may simultaneously analyze frames from multiple cameras.
- Server: Inputs arrive according to a Poisson distribution. This scenario is relevant to online applications where query response times are unpredictable and low latency is crucial. An example of this scenario could be consumer-facing websites that offer translation services.
- Offline: The complete input data set is sent in a unique query. The offline scenario is designed for batch-processing background applications where all data is readily accessible, and latency is not a concern. A suitable example of this would be identifying people and locations in a photo album.

Regarding the benchmarks, MLPerf Inference has two divisions for submitting results: closed and open. Strict rules govern the closed division, such as using specific DNN model implementations, to address the lack of a standard inference-benchmarking workflow. The open division, on the contrary, allows submitters to change the model and demonstrate different performance and quality targets.

As depicted in Figure 3.1, following the MLPerf guidelines and rules to enable ML benchmarking in a serverless environment requires a FaaSification process. FaaSification is the process of transforming existing code into functions in conformance with the programming conventions expected by the target provider. According to Spillner et al. [100], this process can be classified into three levels depending on the considered Atomic Unit (AU): shallow (AU: functions or methods), medium (AU: lines of code) and deep (AU: instructions). This FaaSification process also requires a new scenario for

serverless micro-service architectures. In particular, this scenario executes a burst of several instances with no time interval between consecutive inference queries.

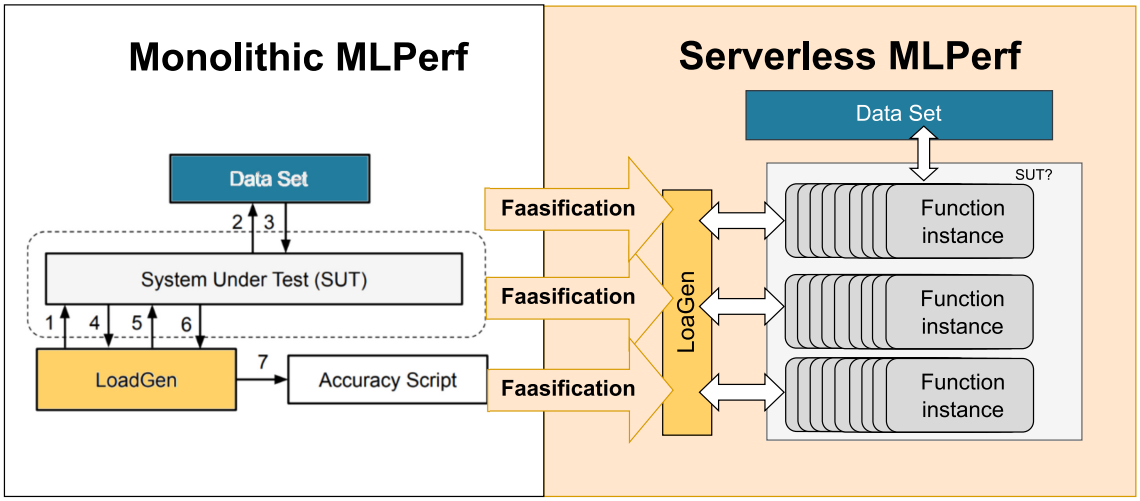


Figure 3.1: The visual representation of the FaaSification process from MLPerf Inference monolithic design [3] to serverless runtime. This process decouples SUT module functionalities into function instances. This process also requires rethinking LoadGen and DataSet interaction with the serverless implementation of the SUT.

Figure 3.2 depicts our FaaS benchmarking architecture and the life cycle of the benchmarking process, numbered from 1 to 11. The SUT is designed following a shallow FaaSification process, i.e., the AUs are functions and methods. More specifically, our SUT implementation is composed of two layers:

- Inference Engine (IE) layer: This layer encompasses the software tools to infer trained DNN models with ML task-related algorithms.
- Handler layer: This layer manages the DNN inference algorithm depending on the selected DNN framework and the post-processing operations to obtain the inference results.

For this FaaS architecture design, we have relied on MLPerf Inference's components, but alternative benchmarking suites could also be considered, as any suite should contain components like SUT, LoadGen and data set.

As shown in Figure 3.2, firstly, the LoadGen configures the data set source from the Online Storage Service (OSS) (step 1). Also, the LoadGen is subscribed to a Notification Service (NS) to handle the benchmarking life cycle (step 2). Next, the warmup

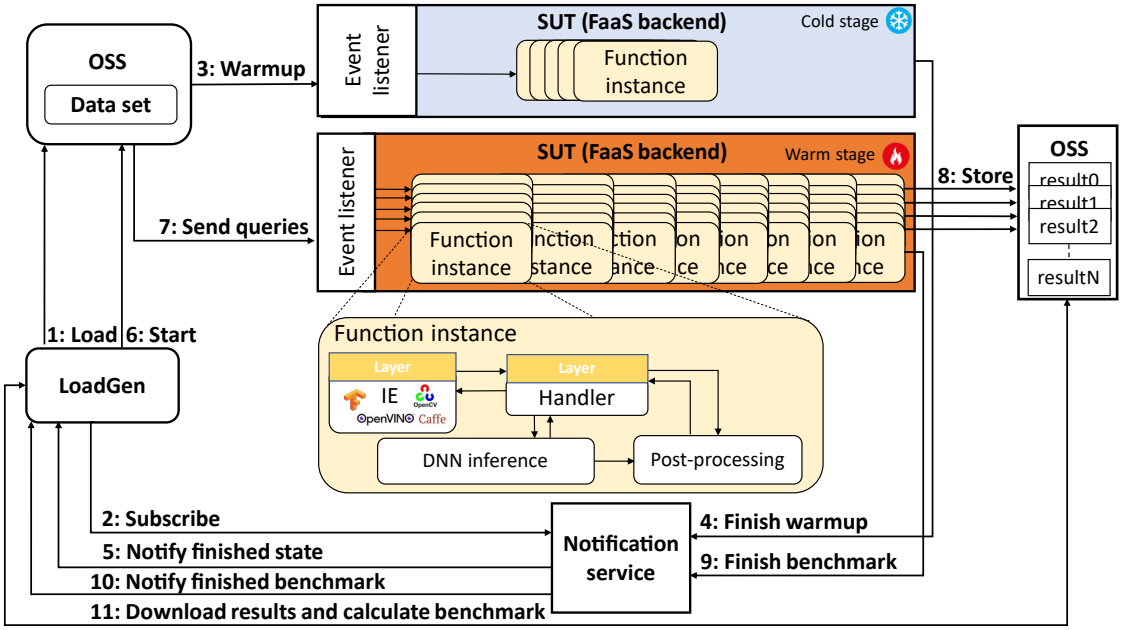


Figure 3.2: The proposed benchmarking architecture for measuring DNN inference performance in serverless environment using MLPerf rules and practical guidelines. The workflow of the benchmarking process is numbered from 1 to 11. This workflow includes data initialization from LoadGen, warmup process for cold start initialization (blue color), and performance evaluation process in the warm stage (red color).

process executes a few function instances to avoid cold start delays during the benchmarking process, and hence, changing the state of the FaaS container to warm stage (step 3). During the first function instance execution, trained DNN models and software tools are downloaded to the containers and then, these DNN resources are loaded to be available for the next warmed function instances. Finally, the LoadGen receives the warmup finish notification from the SUT (steps 4 and 5) and the system is ready to start benchmarking.

For each input element from the data set, the LoadGen uploads a .json file to the OSS (step 6). This file, which contains the paths to the input data set, is comprised of parameters such as database input data references, the result delivering output data, and benchmarking action commands. Next, following the event-driven design of the FaaS platforms [101, 102], each uploading action triggers an event automatically creating a function instance (see event listener in Figure 3.2). At this point in the benchmarking process, the SUT invokes several function instances (one per query). So, each function

performs the inference and post-processing tasks of the data coming from the OSS, and measures the following output values:

- The start and end timestamps.
- The processing time (the function's latency).
- The post-processing results of the function for accuracy evaluation.

To preserve the FaaS data persistence, each function instance saves the measured output values in a separate file in the OSS (step 8). The last function instance sends a finishing action message to the LoadGen (step 9-10). Finally, the LoadGen downloads all files with the mentioned output values from OSS (step 11). This information is organized into different lists to calculate the inference latency, throughput, and accuracy results, which are calculated like this:

- Latency: Instead of using average latency as the definitive metric, the 90th percentile of the latency list is calculated, thus reducing the impact of outliers.
- Throughput: The number of queries divided by the total time. This total time refers to the time difference between the maximum value of the end timestamp and the minimum value of the start timestamp.
- Accuracy calculation: The post-processing results are compared with respect to the ground truth data according to a measurement protocol, which depends on the ML task.

3.2.2 Implementation and evaluation

We tested our approach in Amazon Lambda, with Amazon S3 to store the input and output data, and Amazon Simple Notification Service (SNS) as the NS. As mentioned above, we focused our implementation and tests on computer vision tasks to benchmark the processing capabilities of the DNN inference engines of OpenCV [103] and OpenVINO Inference Engine (IE) [104] using Caffe, Tensorflow, and OpenVINO Intermediate Representation (IR) models. In particular, we have taken the monolithic MLPerf algorithm class and its functions as AUs, and we manually deployed to a FaaS function, supported by Amazon Lambda layers. We have made the source code available

(https://github.com/ Vicomtech/serverless-mlperf) to enable a follow-up discussion about the proposed design, implementation and experiments with the MLPerf and serverless computing communities.

To evaluate the feasibility of our implementation, we benchmarked two DNN models of the MLPerf closed division with the following configuration:

- Data set: ImageNet [60] and COCO [105] (subset of randomized 10K images per data set).
- Performance metrics: Latency and throughput.
- DNN models: We used MobileNetV1 and SSDMobileNetV1 for image classification and object detection respectively, with 32 Floating Point (FP) precision in Caffe, TensorFlow and OpenVINO IR formats.
- FaaS memory configurations: 768MB, 1536MB, 2240MB, 3008MB.

The baseline to compare these results with monolithic implementations would be the results published in the MLPerf's webpage.

Figure 3.3 depicts the latency time for different function memory configurations, from 768MB to 3GB. We observed that BY increasing the allocated memory for each function instance, the latency improves in both benchmarked models, especially in ranges between 768-1536MB. This performance improvement is between 2.12-2.20X larger for MobileNetV1 and 1.58-2.55X for SSDMobileNetV1. This confirms the observations of Maissen et al. [97] about the latency reduction when the allocated memory is increased, and therefore, the CPU power increases linearly.

Moreover, OpenVINO IR models achieve the best performance results. This is because the OpenVINO IE DNN inference engine operations are optimized for Intel parallelization and vectorization instructions such as AVX, SSE2 or SSE4, and currently Amazon Lambda processors rely on Intel hardware [97].

As expected, since SSDMobileNetV1 has more parameters and layers than MobileNetV1, its latency is higher. While in MobileNetV1 the performance of Caffe and TF models is quite similar, in SSDMobileNet the reduction of the latency with the Caffe model is between 1.33X and 1.98X compared to TF.

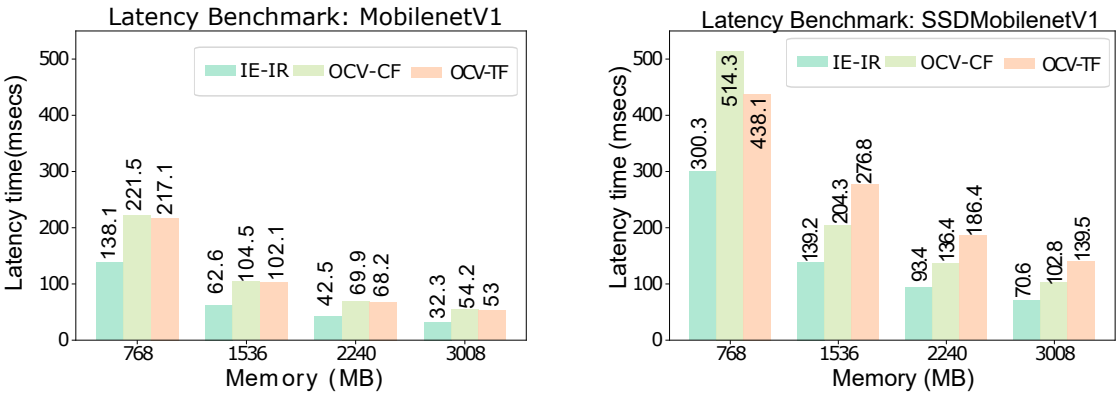


Figure 3.3: Inference latency results with OpenVINO IR (IR), Caffe (CF) and TensorFlow (TF) MobilenetV1 (left) and SSDMobilenetV1 (right) models, and OpenVINO IE (IE) and OpenCV (OCV) as inference engines.

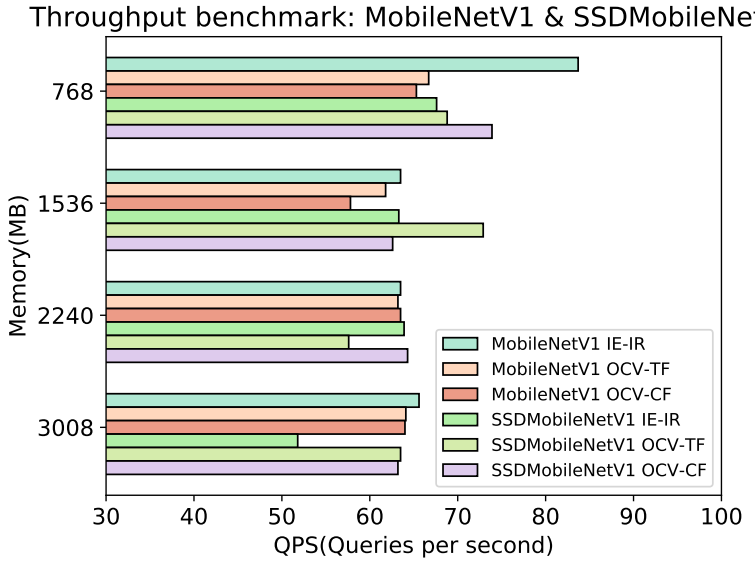


Figure 3.4: Inference throughput results with OpenVINO IR (IR), Caffe (CF) and TensorFlow (TF) DNN models, and OpenVINO IE (IE) and OpenCV (OCV) as inference engines.

Nevertheless, the inference throughput values calculated in Figure 3.4 reveal that the inference latency time does not have any influence in the throughput values. We believe the variations in inference throughput depend on the AWS cloud provider scheduling resources.

3.2.3 Conclusions

The increasing need to deploy ML tasks at high scale demands optimal execution models such as serverless functions. However, finding an efficient DNN inference workload using minimum resources requires an important benchmarking analysis. Throughout our benchmarking evaluation of DNN inference efficiency, we have observed that the amount of the allocated memory for each function instance plays an important role in inference latency time reduction, especially when the configured memory is between 768MB and 1536MB.

Also, the OpenVINO IE DNN inference engine and OpenVINO IR model optimizations contribute to reduce the inference latency in Amazon Lambda hardware. However, these latency results do not influence the inference throughput results. We hypothesize that this occurs due to the cloud provider scheduling capabilities. However, we believe that 51-83 QPS values make Amazon Lambda a suitable platform for DNN inference.

The design space is still very large —different serverless environments, different benchmarks (in addition to MLPerf), different hardware targets (CPUs, GPUs, xPUs, etc)— and requires further investigation. Therefore, we expect to expand these benchmarking evaluations to the most popular serverless function platforms. We will also explore how to benchmark more complex ML systems that consider a computing continuum formed by mobile, edge, and cloud resources [106], relying on standards such as MLPerf.

The previous sections were focused on evaluating the optimal deployment of DNN models in serverless environments, but this analysis is based on a single DNN inference step itself. The DNN-based vision applications, especially for ISVA systems, usually execute person/object tracking including multiple DNNs executing simultaneously. In the following section, we propose a methodology to help build a cost-effective on-demand VSS system, leveraging (1) the latest advances of DNN optimization techniques along with (2) tailored deployment strategies to make the most of current FaaS architectures. This work represents a step forward in distributed computational VSS infrastructures and the VSaaS paradigm [78].

3.3 On-demand serverless video surveillance with optimal deployment of deep neural networks

This section presents a methodology and architecture for deploying multiple DNN models in a serverless environment taking video surveilance as a practical example.

This methodology is comprised of multiple decoupled software layers that allow effectively managing multiple processes, such as business logic, data access, and computer vision algorithms that leverage DNN optimization techniques.

The proposed serverless architecture is illustrated in Figure 3.5, together with the lifecycle of the processing pipeline, where each processing task is numbered from 1 to 11. This pipeline contains two main components: the initialization process (from step 1 to 7) and the on-demand invocation task (from step 8 to 11). The event controller shown in the architecture represents the event-triggering design of FaaS platforms (see Figure 3.5). In this context, each input image source triggers an event for the FaaS function. For security purposes, the images are stored in a Virtual Private Cloud (VPC) and the image data is encrypted. The serverless function is activated by the image itself, avoiding the need for unnecessary read/write permissions on the image data. Following the serverless strategies described above, the designed FaaS function references the resources in the global scope (software layers) and the processing workload relies on the handler function.

3.3.1 Initialization process

When the serverless function is invoked for the first time (cold start), the initialization process begins, and the warm-up process initializes the runtime execution container along with the software layers (step 2-3). Then, DNN models are downloaded to the runtime container (step 5-6). Finally, the DNN models are loaded along with libraries initialization.

FaaS functions should be simple and concise. In addition, FaaS architectures are based on ephemeral storage, which means data will be erased when the function finishes. Based on these requirements, we decoupled the functionalities in three layers which are shared across serverless function instances:

- Deep Learning (DL) layer: In charge of handling DNN workloading operations such as model loading and inference processing, along with pre- and post-processing low-level image operations.
- High Level Algorithm (HLA) layer: Containing the library for complex computer vision pipelines such as face detection, face recognition and body pose detection, supported by several DNN models for inference processing.
- Business Logic (BL) layer: Which provides utilities to deal with accessibility to I/O operations, communications, and business logic algorithms.

Ideally, an optimal initialization process will preserve the accuracy and the inference latency of DNN models [1] under FaaS limitations such as storage size, memory consumption, and computing resources. Model compression techniques, such as pruning and quantization [43], reduce the size of the DNN files, and therefore the required amount of memory to load the compressed model, lessening the cold start delay. These compression techniques are especially relevant when several DNN models are loaded into a layer, because of the rigorous constraint of storage size on FaaS platforms. However, these techniques require special attention in the deployment, since too much compression could dramatically affect the accuracy of the models [68].

Changing the execution runtime from monolithic VSS architecture (all code is processed in the same execution unit) to FaaS architecture requires a FaaSification process. As stated by Spillner et al. [100], this process depends on the Atomic Units (AU), which are classified depending on the level of complexity (third-party dependencies, inter-function dependencies, etc) as shallow (AU: functions or method), medium (AU:lines of code) and deep (AU: instructions). Since the complexity of the DNN processing algorithm lies in three functions (load model, DNN inference, post-processing), our approach is deployed using a shallow FaaSification process supported by HLA and DL layers.

3.3.2 On-demand invocation tasks

After the first invocation of a serverless function instance, the system verifies that all resources are in the warm state and ready for DNN inference. Next, several instances of the handler function are triggered from the input data. These handler instances execute

a set of complex computer vision algorithms which involve multiple DNN inference processing (step 10). Supported by the HLA and DL layer to process vision tasks and DNN inference process, when the FaaS function finishes, the BL layer encodes the algorithm output in the preferred output (step 11).

The serverless platform capabilities to offload the computation across several instances could leverage an impressive DNN inference throughput. However, the virtualized hardware resources of serverless computing become a processing bottleneck, especially, when the vision tasks are required to process many DNN models at the same pipeline.

Based on the analysis of the computational complexity of DNN models [1], choosing the ones that lower inference time while preserving the accuracy is crucial when building this type of serverless architectures. Moreover, vectorization programming libraries such as single input multiple output (SIMD) instructions and multi-threading-block libraries give an extra processing power to the presented serverless architecture.

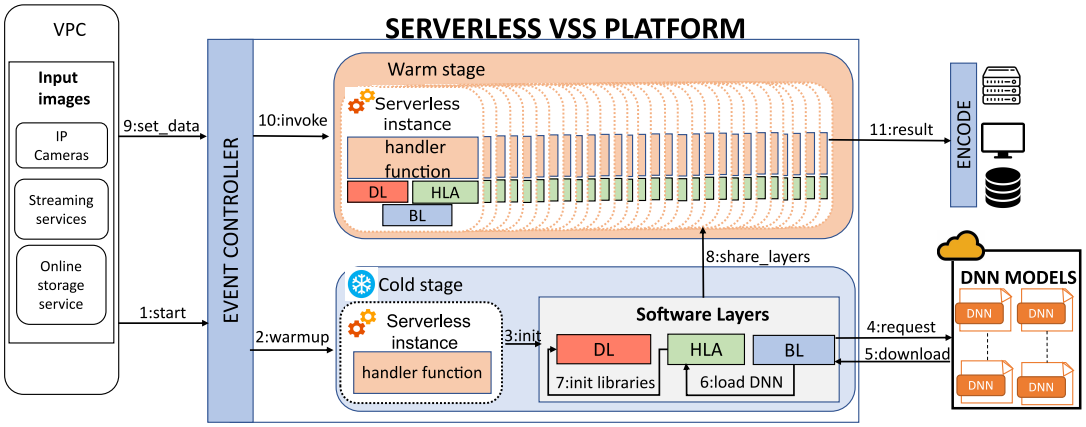


Figure 3.5: The proposed serverless architecture and workflow for deploying multiple DNNs in a video surveillance scenario. The initialization process (cold stage) is represented in blue color while the on-demand execution (warm stage) is in red. The serverless instance comprises three layers: the Deep Learning (DL) layer for DNN inference, the High-Level Algorithm (HLA) layer, and the Business Logic (BL) layer for handling business logic.

The assigned memory to each FaaS function instance plays an important role in performance optimization because, more memory per function means more resources for the handler function, but also higher price per execution. On the contrary, FaaS instances are billed by function execution time, so less time per function means lower

price. Therefore, a good trade-off between allocated memory and function execution cost becomes an essential strategy.

3.3.3 VSS use case and experiments

We evaluated the potential of our approach in the following use case: a VSS that periodically receives (every few minutes) images acquired by several surveillance cameras to detect human presence and recognize registered individuals in uncontrolled environments.

Deep Learning models are complex and require a huge amount of processing power. A Deep Learning model inference lies in matrix multiplications, regularization, and the number of weights. So, choosing the most optimal DNN with minimum latency and maximum accuracy is a crucial strategy. More specifically, in this VSS we deployed four DNNs trained for the following purposes:

- Camera coverage detection (CM): A MobileNetV1-based image classifier to detect whether the image comes from a camera that has been covered (by a hand, a sticker, etc.) or not.
- Human body points detection (HP): An OpenPose-based regression model with MobileNetV1 as the backbone to detect people's body landmarks.
- Human face landmarks detection (FL): A classic convolutional design-based regression model that localizes both eyes, nose tip, and mouth corners in a cropped facial image.
- Human face re-identification (FR): A MobileNetV2-based facial feature extractor for re-identification purposes. The extracted facial features are compared with the registered ones to determine whether they correspond to registered individuals.

Table 3.1 shows the performance parameters of the selected models.

We took AWS Lambda as a baseline to design and test our methodology. The source code is written in Python language. We used OpenVINO as DNN framework and OpenCV for the computer vision algorithms. Also, we used the AWS boto3 library for I/O operations. Since video surveillance environments manage biometric data, to preserve the security of user privacy, we stored all images in a VPC along with an encrypted Amazon

Table 3.1: Selected DNN model parameters for experimental results of section 3.3

NAME	Complexity (GFlops)	AVG Precission (Mp)	AVG Precission(%)
CM	0.569	4.24	70.9
HP	15.435	4.099	42.8
FL	0.021	0.191	92.95
FR	0.588	1.107	99.47

S3 storage service. We also gave the minimum and only necessary permissions to the handler lambda function. Finally, we monitored function calls with Amazon X-Ray. The low-economic impact to process 10,000 images with different batch sizes per request and memory configurations per function is shown in Figure 3.6. Notice the minimum memory to support the VSS application logic is 704MB. The cost calculation is based on the equation 3.1:

$$cost = nr * ((0.0009765625 * am) * (0.001 * ru(rd, m) * mcc + mrc \quad (3.1)$$

Where nr represents the number of requests in a month, am is the allocated memory in MB, rd is the request duration in ms, ru is the round-up operation to the nearest M multiple (m=100 ms), mcc stands for monthly compute charges (0.0000166667 USD/GB-s), and mrc represents monthly request charges (0.0000002 USD/request).

This cost experiment evinces that more images per request involve cost-saving, especially when the allocated memory function is higher. However, despite the cost fluctuation of the first three configurations (704MB to 1536MB) being negligible, the price evolution of the remaining configurations is increased from 13.38% (2048MB) to 61%. (3008MB). Despite the AWS Lambda free tier offering 1 million Amazon Lambda function instances, these function instances are activated with images (put request) coming from S3 online storage service. This S3 free tier offers 2000 put requests (images) in a month. So, for this experiment, the Amazon free tier was discarded because it represented only the 1% of the experiment.

Figures 3.7 and 3.8 analyze the influence of the cold start delays according to local and global scope resource management strategies. In the local scope strategy, all initialization processes are executed in the handler function while the global scope initializes all resources before the first handler function. Each figure contains three different lines which represent the container setup time (green), the runtime init function time (blue),

Figure 3.6: Average cost to process 10K images with VSS in AWS Lambda. The horizontal axis represents image batch size per request (1,5,10) and the color bars represent the allocated memory per function, from 704MB to 3008MB. AWS free tier is not included in this experiment

and the code execution function total time (yellow). The container setup is the time delay to create an isolated image container. In the runtime init function we evaluate the time delay of the serverless function resources (loading external resources, classes initializations, loading third-party libraries, code downloading). Finally, the function code execution calculates the total execution time of a serverless cold instance.

Our analysis of the resource management strategies reveals that initializing the resources in the global scope improves the performance of the cold start delays (about 2-4 seconds difference). Also, the increased time delay of the global scope in function runtime initialization is due to the DNN models, code, and libraries loaded in this step. As it was expected, as far as the allocated memory per function instance increases, the cold-start time delay is reduced in both scope strategies. This time reduction is especially visible when the allocated memory is between 704 and 1536MB. In contrast, the container setup's minimal time variations reveal that the FaaS container initialization does not depend on the allocated memory per function instance. To analyze the cost-worthiness of serverless computing deployment, Figure 3.9 reveals that our FaaS architecture leverages an outstanding performance with an important time saving from hours, which would be needed with an off-the-shelf PC, to minutes (our ap-

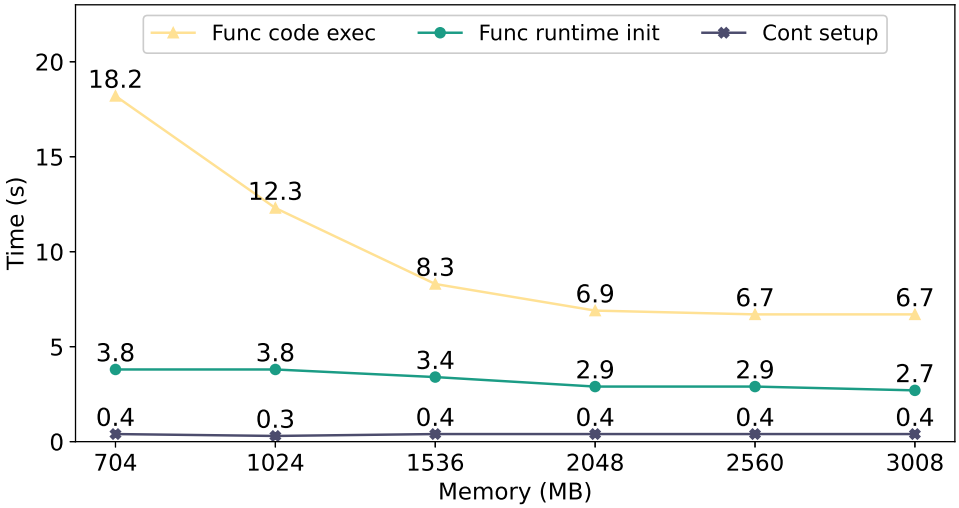


Figure 3.7: Cold start time analysis of the global scope strategy according to the amount of allocated memory per function (from 704MB to 3008MB).

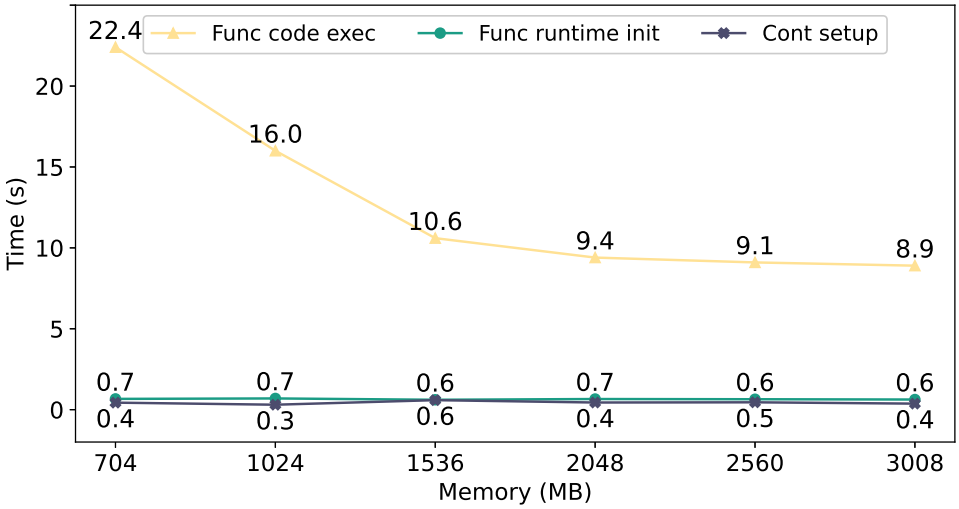


Figure 3.8: Cold start time analysis of the local scope strategy according to the amount of allocated memory per function (from 704MB to 3008MB).

proach). Also, the influence of the allocated memory per function instance is shown in Figure 3.7, where the reduction of the processing time is very significant, especially between the 704MB and 1536MB configurations. Considering the economic and the time

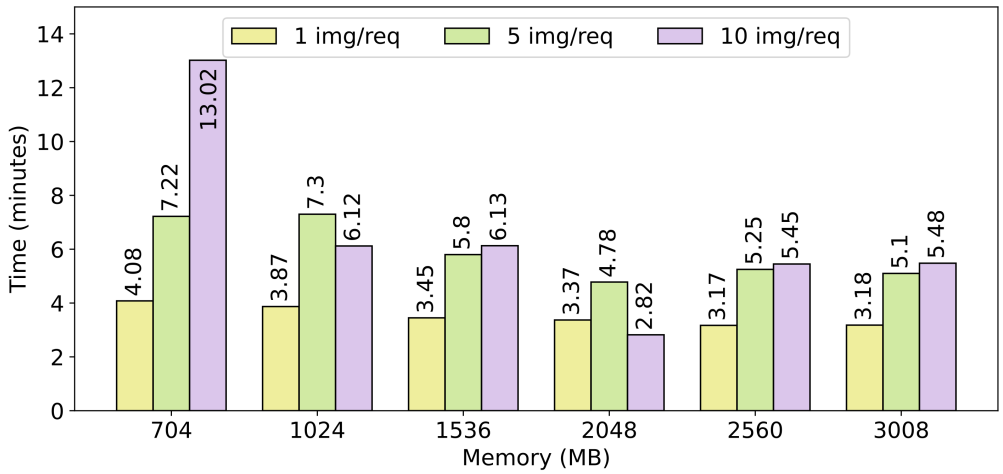


Figure 3.9: Total times to process 10K images with the VSS in AWS-Lambda. The color bars represent image batch size.

performance analysis shown in Figures 3.6, 3.7, and 3.8, we conclude that the optimal allocated memory per function remains on 1536MB. Also, as observed in Figure 3.9, the optimal way to achieve the maximum processing throughput is processing one image per each FaaS instance.

3.3.4 Conclusions

The FaaS platform environment offers a suitable distributed execution model to provide parallel processing at a high scale. Nevertheless, the resource limitations of this platform collide with the DNN complex environment. To overcome this challenge, we have presented a methodology to optimally deploy several DNN models to FaaS platforms supported by the latest computer vision techniques to maximize the DNN processing performance at minimum cost. We have also evaluated a VSS case study supported by experimental results that reveal an outstanding performance improvement of our serverless architecture. Furthermore, we conclude that the major bottleneck lies in the processing of each FaaS function, while the influence of the memory allocation per function is visible in the processing speed. Nevertheless, there is large room for improvement in reducing the DNN complex environment, while the bottleneck could be addressed by analyzing the possibilities of distributing the DNN processing into

multi-tenant systems.

Current computing requirements for high-scale inference of DNNs demand distributed execution environments. The advantages of serverless functions in distributed computation offloading and automatic resource scalability make them a very suitable environment for such a task.

CHAPTER

4

Knowledge-driven approach for the optimal deployment of DNNs in heterogeneous IoT platforms

This chapter explores the optimal deployment of DNNs in heterogeneous IoT platforms by means of a knowledge-driven approach. More specifically, our focus is on addressing the deployment challenges of face recognition solutions in IoT platforms for older adult care and industrial applications. We analyze various approaches and propose a knowledge-driven methodology for automated deployment of DNN-based face recognition solutions on IoT devices. This approach manages biometric data securely and provides real-time feedback for improved user interaction. This chapter refers to contribution 4 described in Section 1.3.

The chapter is organized into five sections. Section 4.1 describes the background and challenges for face recognition-based user authentication in heterogeneous IoT platforms. Section 4.2 defines the whole face recognition methodology including user interaction workflow (subsection 4.2.1), the automatic deployment process of face verification algorithms(subsection 4.2.2), and the secure biometric data management

process (subsection 4.2.3). Section 4.3 presents the results and evaluation of our proposed methodology. Section 4.4 provides practical examples of how the proposed approach can be applied in different kinds of devices, including IoT gateways, devices without DNN inference capabilities, and devices that process the face recognition pipeline locally at the edge. Finally, in Section 4.5, we draw conclusions and suggest ideas for future research.

4.1 Background and challenges

Industry 4.0, the fourth Industrial Revolution, is focusing on interconnectivity, automation, autonomy, machine learning, and real-time data, in parallel to the growing number of interconnected devices around the world [107]. The interconnected devices could be very diverse: wearables, smart appliances, smartphones, tablets, smart TVs, embedded computers, gateways, laptops, computers, different kinds of robots, etc. In this context, the IoT paradigm plays a major role, using applications running on devices with sensing, networking, and processing capabilities that interact with other devices and services on the Internet. IoT allows integrating the physical world into computer-based systems, providing manufacturing companies with many growth opportunities. Thus, expanding IoT capabilities will enable more sophisticated products and services to contribute to the progress of Industry 4.0. One way to expand them is to improve the sensing capabilities of interconnected IoT devices to detect and recognize users, to allow them to interact securely with IoT applications.

In recent years, DNNs have allowed the development of robust solutions for computer vision tasks such as face recognition, including face detection, alignment, and identity recognition [108] [109] [110]. Current face recognition solutions can help to secure the use of IoT applications, by restricting the access to sensitive data to individuals with the required permissions, in a pervasive and user-friendly way, without the need to have something (e.g., PINs or passwords) to memorize. Users could perform authentication by having their face recognized on IoT devices with cameras, and this one-time login could provide access to a full IoT network of devices of many different kinds. Furthermore, some applications might require these machines to identify people at a distance without their collaboration (e.g., to localize users, or for surveillance

purposes), and face recognition opens up this possibility, unlike other biometric-based authentication alternatives, such as fingerprints, iris, or hand geometry [111].

However, deploying DNN-based face recognition solutions in IoT platforms is a challenging problem mainly due to the following factors:

- Taking into account that to obtain (near) real-time responses, DNN models need to be processed locally, and not on remote servers—as server-device data transference would add considerable latency in such cases—the computational cost of DNN inference could be higher than the computational resources available in many IoT devices. In addition, they could have different kinds of processors (xPUs: CPUs, GPUs, FPGAs, etc.), which require specific DNN inference engines (Intel's OpenVINO, Google's TensorFlow Lite, NVidia's TensorRT, Facebook's PyTorch, etc) [3].

- To allow users to enroll on one device and authenticate on another, respecting their privacy in compliance with the law, such as the EU's General Data Protection Regulation (GDPR), biometric data needs to be managed securely, preventing intruders from accessing it.

- Besides the high heterogeneity of IoT devices with which users might interact in terms of shape, functionalities, sensing, and computing capabilities, we might face a high variety of user interaction capabilities, from fully active to fully assisted (e.g., in older adult care applications). All users should be able to interact satisfactorily with the deployed face recognition system during the face enrollment and verification stages.

In the upcoming sections, we address these three challenges with a focus on older adult care and industrial applications.

4.2 Methodology

4.2.1 User interaction workflow for face verification

We propose the user interaction and recognition workflow shown in Figure 4.1. It considers the user-centered design principles described in [112], and the importance of

providing real-time feedback, as suggested in [113], not only to obtain good quality images but also to improve the technology acceptance and adoption. Thus, the user will trigger this workflow the first time to enroll into the system, and every following time there is the need to verify the user's identity.

The workflow is divided into three phases: (1) facial image acquisition, (2) spoof detection, and (3) biometric features extraction. The goal of phase 1 is to assist the user in presenting themselves appropriately to the camera to extract facial images with sufficient quality for the subsequent phases. Then, in phase 2, the system will check that there is not an attempt of spoofing, and if there is, it would ask for a new image, going back to phase 1. Finally, in phase 3, it will perform the biometric feature extraction for the registration or the verification. Some users could have physical difficulties that could interfere with the interaction with the device, making it difficult to hold it still when taking a picture. For this reason, the proposed interaction does not present a button to press when taking the image, as this action could cause extra movement and result in a blurred image. The system will automatically take the image when all the required quality conditions are met.

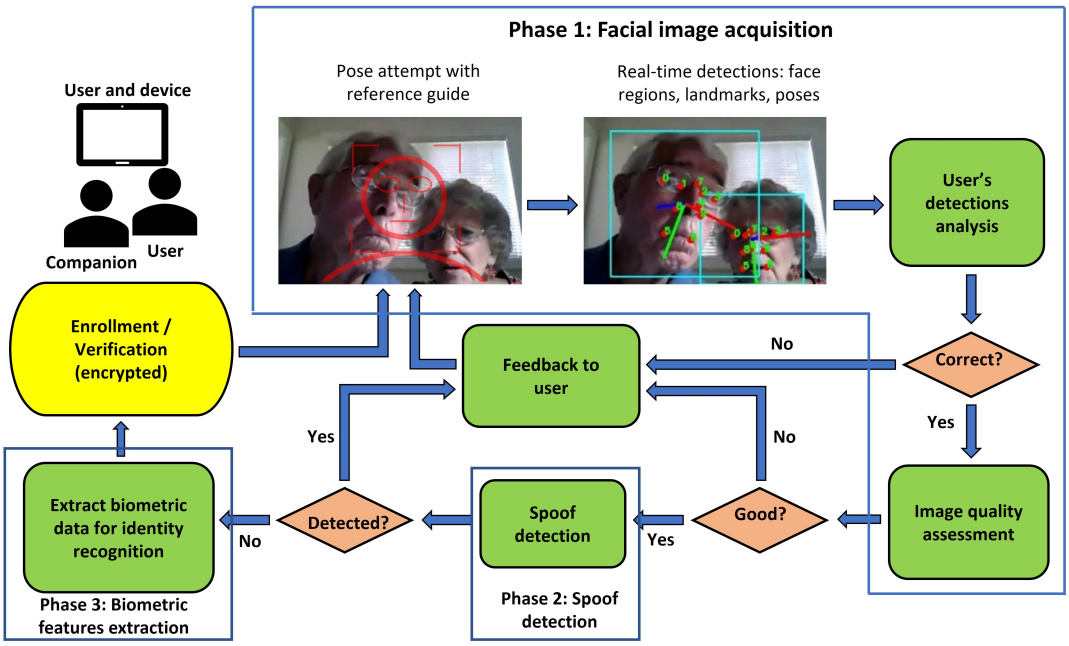


Figure 4.1: User interaction and face recognition workflow in different phases. (1) facial image acquisition, (2) spoofing detection, (3) biometric feature extraction.

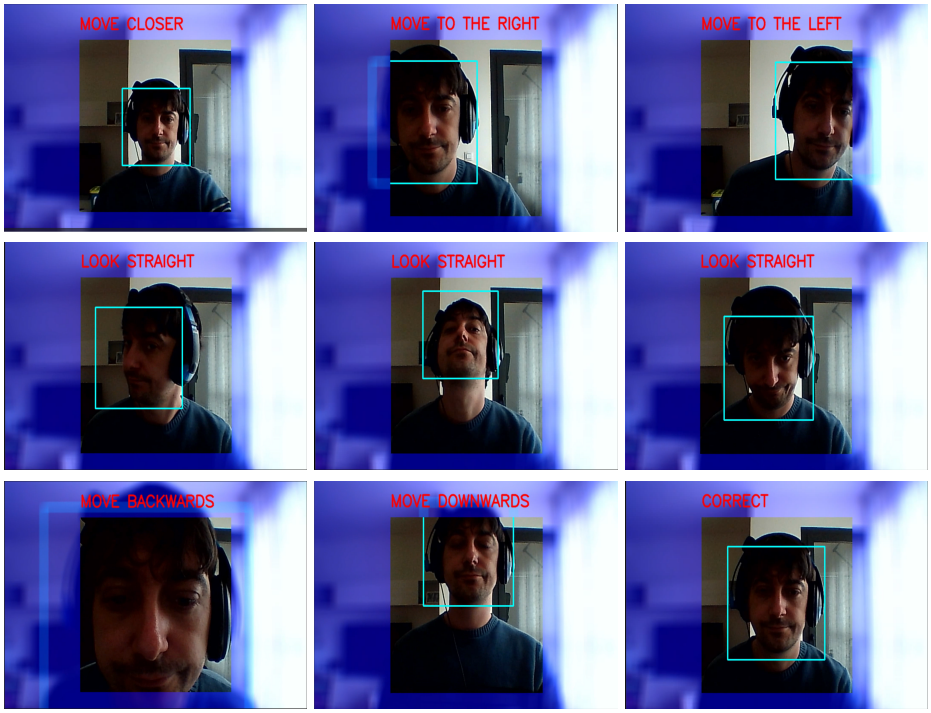


Figure 4.2: Example of a user interacting with the face recognition application and receiving feedback in real-time during the image acquisition process.

More specifically, in phase 1, the system's interface will show the mirrored video stream captured by the device's camera with an overlapping human shape that will work as a guideline for users. First, based on the face regions, facial landmarks, head pose, and facial gesture estimations obtained by a series of DNNs [114] [115], the system will indicate if the face is too close or far from the camera ("move closer", "move further"), or if the head is tilted ("move to the left" or "move to the right"). Since some users might have difficulties in reading small text, an icon would indicate the movement to facilitate the comprehension of the feedback messages. In the case the user is accompanied by someone else (e.g., an assistant), the system will detect more than one face, but it will consider only the face closer to the device as the user that requires verification, hence providing the feedback to that specific user. Figure 4.2 shows some examples of the user interface and the provided feedback.

Then, once the image passes this check, the user's facial image is cropped and "normalized" based on the detected facial landmarks (i.e., it is resized and rotated so that the eyes are in predefined positions in the cropped image).

This normalized facial image is then processed by another DNN for face image quality (FIQ) assessment [116]. FIQ methodologies provide a score that can also be obtained to check whether the image respects the required conditions, and, providing feedback to improve when they are not met (with a message like "improve the lighting conditions"). This quality score is defined as FIQ value. This proposed face recognition workflow includes FIQ assessment methodology based on multi-task explainable quality networks (XQNets) [29] for large-scale forensic face recognition applications. More specifically, this network allows assessing FIQ efficiently along with a set of environmental variables that explain the calculated FIQ value. This way, the feedback provided to the user when posing in front of the camera can be more sophisticated to correct the environmental conditions if required. Apart from the FIQ overall information, this model also offers the following information: hot spots, sharpness, and deviation from uniform lighting.

If this quality check is also passed, the processing of the user's normalized facial image continues in phase 2. During phases 2 and 3 there is no need to show the user the mirror image from the camera as it could be confusing (seeing the image the whole time they might think that there is something wrong with the image acquisition when it has been acquired already). In phase 2, following the observations made in [117], to improve the security of the verification system, another DNN would verify whether the normalized facial image corresponds to a spoofing attack or not [118] [119]. In our implementation for the spoof detection procedure, we used a MobileNetV3-based classification model, trained with the Celeb-Spoof dataset [119] due to its lightweight processing requirements and high accuracy processing capabilities. The anti-spoofing procedure could be further enhanced by using depth sensors in addition to a camera. Figure 4.3 shows examples of FIQ assessment and anti-spoofing results in various scenarios, including different identification card types, mobile-reproduced videos, and wearing masks.

Finally, in phase 3, another DNN would extract the biometric feature vector (i.e., the i-vector) from the normalized facial image that allows authenticating facial identities [120]. Once the features are extracted, the system can proceed to compare them against those stored during registration with a verification algorithm -that could involve Euclidean distance, cosine similarity, etc.—and grant the access or not depending on the result.

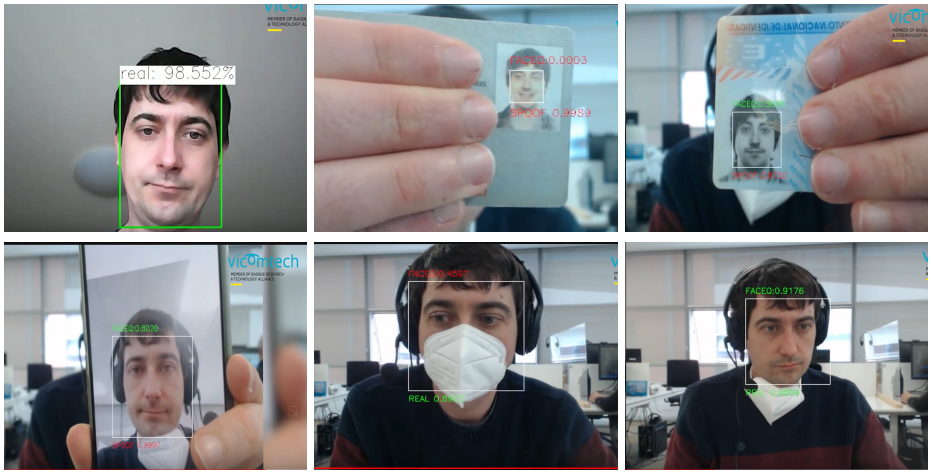


Figure 4.3: Examples of results from the FIQ and anti-spoofing algorithms for various situations and spoofing attacks using mobile and webcam cameras.

In order to preserve the user's privacy, no facial images are stored, neither during the user registration process nor during the surveillance/authentication process, only the extracted i-vectors are used, which are encrypted and managed as explained in Section 4.2.3 to preserve the user's privacy.

4.2.2 Deployment of face recognition algorithms

Figure 4.4 shows our proposed knowledge-driven workflow for the automated deployment of face recognition solutions on heterogeneous IoT platforms for industrial applications (e.g., on the private network of a company's facilities). This deployment scenario considers a wide range of devices and robots with whom users might interact. To reduce the latency of the face recognition solution as much as possible, this deployment workflow considers two runtime scenarios. The first scenario contemplates on-device DNN inference for devices with DNN inference capabilities. The second scenario deploys the face recognition solution on the IoT gateway to handle multiple requests at once using containerized services. This deployment process categorizes the client devices and robots into suitable and not suitable devices for DNN inference. Those suitable will host the face recognition service directly, and the rest will communicate with the IoT gateway when required, by submitting images and processing requests to the gateway and receiving the corresponding responses. To improve the user experi-

ence, users register their face credentials on one device and the biometric data is shared among the rest of the private network's IoT devices. Thus, the face login can be done on any of these devices. Even though this network is private, biometric data encryption is necessary, so the administrator manages the encryption keys to preserve privacy, as explained in the next section.

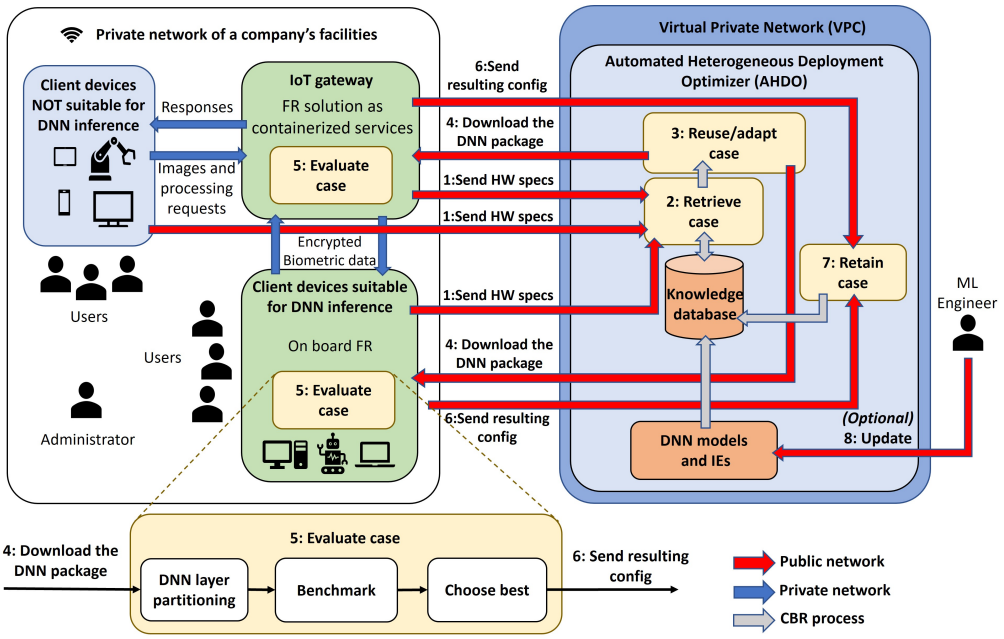


Figure 4.4: Workflow for the automated deployment of the IoT face recognition solution.

This deployment procedure automatically decides which is the most suitable DNN package—DNN models and Inference Engine (IE)—for optimal DNN inference in each IoT device. The decisions are taken by a case-based reasoning (CBR) system [115]. A CBR is a problem-solving method where the solutions are based on previous experiences. A CBR system is organized in cases, which are represented as problems and solutions. An "experience" instance would correspond to a previously solved case. In our context, we have one case type that would be represented like this:

- Problem: New device with heterogeneous hardware (i.e., might have one or more kinds of processors, including in some cases DNN accelerators).
- Solution: The most optimal DNN IE and DNN model configuration package for the target device.

As it is shown in Figure 4.4, the automated deployment workflow has 8 steps, and the CBR process is its core. The standard CBR process is comprised of four tasks: (1) Retrieve case, (2) reuse/adapt case, (3) evaluate case, and (4) retain case. Almost all tasks are executed in a module called automated heterogeneous deployment optimizer (AHDO) which is hosted on a virtual private cloud (VPC). In contrast, each IoT device and gateways with DNN inference capabilities execute the case evaluation task. Next, we will explain these steps and components in detail.

4.2.2.1 **Case retrieval**

This task is the CBR's starting point. Its goal is to retrieve the most similar old cases (experiences) to the new case. Each client and the IoT gateway activate this retrieving task sending their hardware specifications to the AHDO (step 1). Each hardware specification represents a new case for the CBR system. The old cases are stored in the knowledge database. All cases (new, old) follow a specific case structure which we define in categorical and quantitative values. The categorical values represent the hardware vendor (Intel, NVidia, Google), the DNN accelerator types (GPU, CPU, VPU, TPU, FPGA, etc.), and system architecture (x86, arm64, etc.). The quantitative values represent a feature vector of hardware characteristics such as number of cores, clock speed, and dedicated RAM.

This case retrieving task analyzes the categorical and the quantitative similarities between old and new cases (step 2), as shown in Figure 4.5. First, it calculates the categorical similarity using the Hamming distance. This operation returns the most similar results (the lower the distance, the more similar) as a list of tuples associated to a tuple identifier. Those with a distance value lower than the threshold cat_thr are selected for the quantitative similarity measurement. From this list, the system takes the tuple identifiers and extracts the quantitative values of each DNN accelerator, represented as feature vectors. Then, the cosine similarities between each feature vector and the new case feature vector are calculated. From these, if the highest value is higher than the threshold quant_thr, the solution is found (case reuse), otherwise the solution requires an adaptation (case adaptation). When the knowledge database is empty, the new case goes to the case adaptation step, automatically.

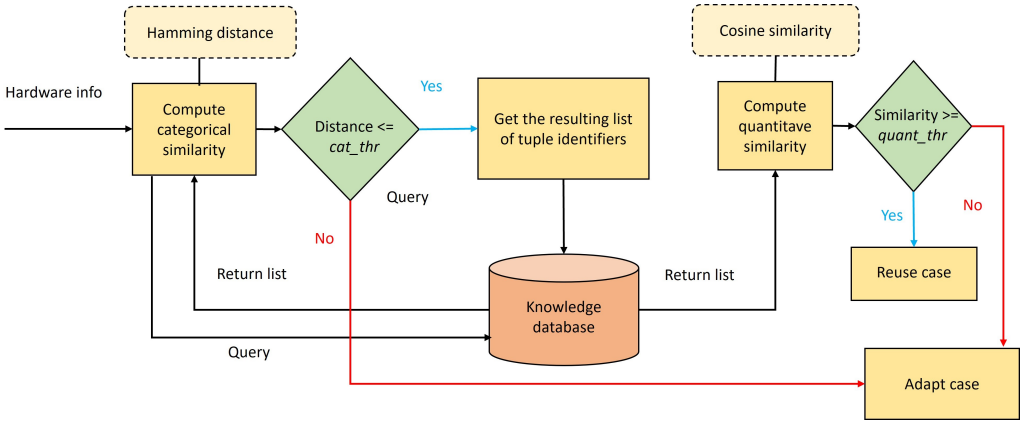


Figure 4.5: Case retrieving workflow for knowledge-driven methodology.

4.2.2.2 **Case reuse/adaptation**

As mentioned in the previous section, this task finds the solution of a problem providing two alternatives, the reuse of a retrieved case solution or the adaptation of the new case solution (step 3). The case reuse only associates the previous retrieved solution to the new case problem. The case adaptation instead involves a more complex task. However, for both cases, the VPC creates a unified package of candidate DNN resources (DNN IEs, DNN models, and also a small dataset of labeled images for testing) and transfers it to the target device, robot, or gateway (step 4).

For the case adaptation, we choose a rule-based engine to find the solution to the new case. We define a set of rules which represents the relationship between DNN IEs and DNN models. These rules are defined by an expert human (Machine Learning engineer). These defined rules and the hardware information of the new case are executed in the rule-based engine (expert system) and as a result, the expert system returns a list of possible candidates of DNN IE and DNN models, along with the testing dataset.

These DNN models can be of the same kind but trained in different DNN IEs and with different precisions (from most accurate to fastest: FP32, FP16, or INT8). Some DNN IEs require using specific computing processors (e.g., Google's TFLite Edge TPU for Google's TPUs, or NVidia's TensorRT for NVidia's GPUs), but in other cases, the same processor is suitable for different DNN IEs. Moreover, the face recognition solution is composed of several DNNs such as face and face-landmark detection, head pose, facial gesture, facial image quality analyzer, anti-spoofing detector, and identity recognition.

These models can be deployed into heterogeneous hardware. For example, some models could run in one CPU processor and the remaining models in the GPU processor, depending on the requests of image batches that suit best with the processor. Here, with batch processing, we do not refer to resizing the input of the DNN model to a specific image batch size, but to a batch of several inference requests which are executed asynchronously. Depending on the target hardware architecture, optimizing the inference process such as splitting the N batch of images in N requests or using inference asynchronously could improve the performance. That is why choosing the optimal model configuration plays an essential role in this deployment process.

4.2.2.3 **Case evaluation**

In step 5, the CBR system evaluates the solution proposed in the previous case reuse/adaptation task. More specifically, it evaluates the performance of the solution candidates of DNN IE and models. This task is done directly on the IoT device. This evaluation process, apart from benchmarking the inference latency and accuracy of the DNN models, also analyzes the DNN graph partitioning capabilities for those models in the DNN IEs that have heterogeneous inference capabilities, such as OpenVINO and TensorRT [121]. The DNN graph partitioning consists of splitting the DNN model into different subgraphs (group of layers) and delegating their workload to the most suitable DNN accelerators. The case evaluation is shown in the algorithm 1.

This algorithm returns a list of optimal hardware configurations (OHC) for each batch size. The algorithm iterates through all possible batch sizes and, for each one, iterates through their corresponding heterogeneous hardware configuration (HCONF). For each pair of batches and downloaded configuration from the VPC, the algorithm loads the corresponding IE based on the hardware heterogeneous configuration and selects the required precision model according to the VPC downloaded candidate. Then, the algorithm chooses a suitable DNN model for benchmarking and the image testing database included in the VPC downloaded package. When the selected model is loaded, the algorithm checks if all the DNN layers are supported by the device. If all layers of the DNN model are suitable for the current DNN IE, the benchmark is performed, otherwise, the configuration is discarded.

Algorithm 1 Case evaluation algorithm

```
1: procedure EVALCASE(HCONF ,IB ,DMC ,DNNIE , TESTDATA )
Input: HCONF                                   ▷ Heterogenous HW device conf
2: IB                                          ▷ Image/request inference batch sizes list
3: DMC                                         ▷ DNN model candidates
4: DNNIE                                       ▷ DNN IE
5: TESTDATA                                    ▷ Testing dataset for benchmarking
Output: OHC                                    ▷ List of heterogeneous HW conf per batch
6:     for batch in IB do
7:         for hdconf in HCONF do
8:             for ie in DNNIE do
9:                 dm = getPrecisionDNNModels(ie, DMC,hdconf)
10:                db=loadDatatForBench(TESTDATA)
11:                loadModelsToIE(ie, dm, db,hdconf )
12:                aff=getLayerAffinities(ie,dm,hdconf)
13:                if allLayersSupported(ie, hdconf,dm) then
14:                    makeBench(ie, dm,hdconf,aff, db)
15:                    perflistdev=storeBenchMetrics(dm,hdconf,ib)
16:                    optconfb = findOptimalConf(perflistdev,ib)
17:                    OHC.append(optconfb)
18:                else
19:                    discardDevConf(dm, hdconf)
                   Return OHC
```

When the DNN graph partitioning is not supported by the device, this step is skipped, but the benchmarking is executed anyway for the evaluation process. Finally, for each batch size, the findOptimalConf function selects the best DNN model configuration finding the trade-off between inference latency and accuracy results following the MLPerf benchmarking standard [3]. Then this configuration is added to the OHC list.

4.2.2.4 **Case retaining**

Steps 6 and 7 are undertaken to submit the selected configuration back to the VPC and then, this new case (problem and solution) is stored in the knowledge database —the case problem (categorical and quantitative values) and the case solution (the most optimal DNN IE and DNN model configuration).

4.2.2.5 **Updating the new trained DNN models**

Since the DNN models and DNN IEs are constantly evolving, we consider the updating process of these resources (step 8). This process not only consist of uploading DNN resources, but also the rules and relations for the rule-based engine. Thus, these new rules, DNN resources are stored in the knowledge database. This updating process also requires evaluating if these new DNN resources are more suitable for the IoT device. Thus, when the updating process is finished, the VPC notifies to all IoT devices that the new DNN resources are available.

4.2.2.6 **Running the face recognition system**

When the system is ready to load the face recognition system, our approach provides the functionality to load the most optimal model. But when the DNN model is loaded to the DNN accelerator memory, its graph cannot change during runtime. The model's performance depends on several factors such as its case scenario, input data throughput (number of face images to be processed simultaneously) and batch size, which leads to situations where models with larger batch sizes may not obtain the best results. In cases where several faces need to be verified configurations with larger batch sizes perform faster than other options with the highest precision models and the face recognition system's workload will use the 100 % of the hardware resources, the most optimal way to use them. In the opposite situation, when the input face images throughput is low, choosing this option could lead to energy waste, degrading their performance. Furthermore, choosing cheaper hardware accelerators and reducing batch size can perform similarly with lower power consumption.

4.2.3 **Biometric data management**

The biometric data of the user results from processing the users' facial images with a DNN trained with data that correspond to other people, but not to the user. Thus, this biometric data is an abstract representation generated by a combination of the appearances of some of the people used for training the DNN, but not specifically from the user. This means that it would be impossible to reconstruct with high precision the user's real appearance from the biometric data, using techniques such as those described in [122]. Nevertheless, this kind of reconstruction could reveal relevant personal

information from the real user, due to the similarity to those people, such as “gender”, “skin color”, “hairstyle”, etc., which needs to be protected. While a completely secure system against any kind of threats does not exist, an appropriate level of security for the targeted scenario and the expected kind of attacks can be designed.

In this approach, biometric data would never be transferred to the cloud, i.e., it would only be stored in those devices where the face recognition solution has been deployed. Thus, an expected possible attack could be somebody accessing the devices, robots, or gateway to steal the stored data. To prevent this, the data should be encrypted, and the encryption keys should be kept safe by an administrator.

In our context, we consider the method [123] as the most appropriate for our purpose. It proposes an efficient fully homomorphic encryption-based approach to cryptographically secure the registered and the probe biometric data, performing the matching directly in the encrypted domain, leveraging the observation that a typical face matching metric, either Euclidean distance or cosine similarity, can be decomposed into its constituent series of addition and multiplication operations. It utilizes a batching scheme that allows homomorphic multiplication of multiple values at the cost of a single homomorphic multiplication, and dimensionality reduction to further provide a trade-off between computational efficiency and matching performance. Therefore, the biometric data shared among the devices, robots, and gateway will always remain encrypted, as long as the administrator keeps the encryption keys safe (e.g., in a secure hardware element [124]).

This approach requires 4 keys: a public key for encryption, a private key for decryption of the scores, and linearization and Galois keys for the matching operations of the encrypted data (Figure 4.6). This way, if someone steals the stored biometric data from the hardware platform, it would not be usable without access to the private key required for decryption.

This private key should be kept safely in a secure element of the hardware platform, for example, a trusted platform module (TPM) or a trusted execution environment (TEE). TPMs are normally available in modern computer PC motherboards. TEEs are available in Jetson Xavier NX, Jetson AGX Xavier series, and Jetson TX2 series devices.

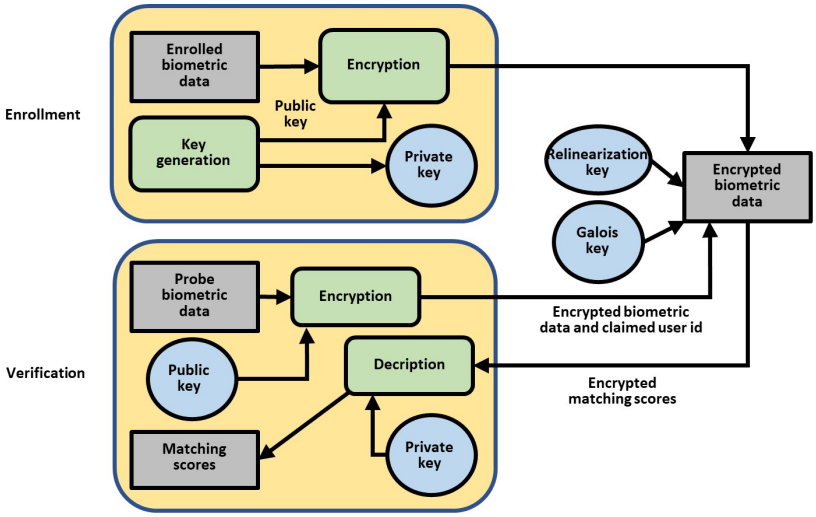


Figure 4.6: Biometric data management with fully homomorphic encryption during enrollment and verification.

4.3 Experiments and discussion

4.3.1 Qualitative evaluation

Table 4.1 shows a qualitative comparison of our approach with respect to state-of-the-art alternatives, in terms of face recognition technology, interaction assistance, deployment algorithms and privacy. We also include in this comparison our previous version [27] to see the differences more in detail. As can be observed, both [27] and the improved approach presented in this paper stand out in terms of interaction assistance and deployment, issues which were poorly addressed by the other alternatives. The main differences between [27] and the extended version are the inclusion of a DNN for image quality assessment, which allows an improved interaction with the user, and especially, the way in which the DNN resources are deployed, by means of the presented knowledge-driven approach, which takes advantage of previous cases (experience) to reuse and adapt the knowledge to new cases, as required. The versatility offered by our deployment scheme overcomes previous approaches by removing dependence on the devices' hardware and allowing usage of more complex DNN models with improved face recognition capabilities. In terms of privacy and data protection, our method employs an efficient fully homomorphic encryption-based approach and restricts all operations to local devices, offering a lightweight processing pipeline without compromising se-

curity. All these features make our approach well-suited to all demands required by an industrial scenario to be deployed among workers.

Table 4.1: Comparison of state-of-the-art IoT platform approaches vs. our proposal (FD: Face detection, FLD: Face and facial landmark detection, PGR: Pose and gesture recognition, IQA: image quality analysis, SAD: spoofing attack detection, FIR: Facial identity recognition).

Method	Face recognition approach	Assistance interaction	Deployment of algorithms	Privacy in the IoT platform
[92] [93]	Pretrained Haar-based for FD and LBP features model for FIR to be trained on the cloud.	Not considered	Manually predefined	Schemes for authentication, session key agreement, integrity checking for secure data encryption, and data transmission and storage
[94]	Discriminative dictionary learning for FIR that needs to be trained	Not Considered	Manually predefined	Biometric data encrypted with a low complexity encrypting algorithm based on random unitary transformation
[95]	DNN for FIR split in two parts: one deployed on the user side and the other on the edge server side.	Not considered	Manually predefined	Differential privacy for user's confidential datasets. No cryptographic tools used to keep user side lightweight.
[27]	Pretrained DNN models for FLD, PGR, SAD, deployed on fog gateway and client devices suitable for DNN inference.	Real-time visual feedback based on FLD and PGR to guide the user during enrollment and verification.	Automated selection of the appropriate DNN inference engine DNN model configurations and, batch size, based on . IoT device characteristics	Biometric data homomorphically encrypted. All computations are done on the private network. Biometric data not sent to the cloud
Ours	Pretrained DNN models for FLD, PGR, IQA, SAD, and FIR deployed on fog gateway and client devices suitable for DNN inference.	Real-time visual feedback based on FLD, PGR and IQA to guide the user during enrollment and verification.	Automated selection of the appropriate, DNN inference engine, DNN model configurations, batch size, by means of a, knowledge-driven approach.	Biometric data homomorphicall encrypted. All computations are done on the private network. Biometric data not sent to the cloud

4.3.2 **Quantitative Evaluation**

In order to evaluate the performance that a DNN-based face recognition solution would achieve following our deployment approach, compared to standard—manually predefined—deployments, we completed two experiments on two different target devices. We used MobileNetV1 [33], ResNet-50 [51], and SSD-MobileNetV1 [59], which are popular DNN architectures for the computer vision tasks of "image classification" (for instance, all the tasks applied to cropped facial images; spoofing detection, identity recognition,

etc), and "object detection" (face regions in our case). The weights of these DNNs have FP16 precision. We used the following hardware for the benchmarking experiments:

- IEI TANK AIoT Developer Kit embedded PC with Intel Mustang V100 MX8 for DNN acceleration card. This hardware contains an Intel CPU, GPU and a High Density Deep Learning (HDDL) card (Mustang) compatible with Intel's OpenVINO DNN IE.

- NVidia Jetson Xavier AGX 32GB. This hardware contains a NVidia GPU with 512-core NVidia Volta GPU with 64 Tensor cores and 2 NVDLA (dla0, dla1) DNN accelerators compatible with NVidia's TensorRT DNN IE.

The first experiment consisted of measuring the performance of MobileNetV1 and ResNet-50 for different image batch sizes (1 to 48) with different inference configurations, assuming that all hardware resources are fully available. These two hardware platforms allow DNN graph partitioning across heterogeneous hardware, given a selection of computing processors and their execution priority. For the TANK we considered the following configurations: <CPU, HDDL>, <GPU, CPU> and <HDDL, GPU, CPU>, where the order represents the execution priority. For the Jetson we se-lected the following configurations: <GPU>, <DLA0, GPU>, <DLA1, GPU>. The current version of Jetpack (4.5.1) —the Jetson's SDK—, does not allow the deployment of a DNN model across both DLA cores (DLA0 and DLA1). Each DLA core is totally independent and can only communicate with the GPU for the inference.

Figure 4.7 shows that the <GPU, CPU> configuration performs best with smaller batches (1 to 4), while <HDDL, GPU, CPU> is the fastest for higher batch sizes. Those would be the configurations chosen by our approach for those batch sizes (the optimal). In principle, the CPU's and GPU's hardware processing capabilities are bigger than those of the Mustang V100. Thus, theoretically the <GPU, CPU> configuration should be the best choice for all batch size configurations. However, the parallelization processing methods of the CPU, based on SIMD vectorization techniques, are designed for a general parallelization purpose, and not for DNN inference processing. Also, the cache L1/L2/L3 memory is shared across the CPU and GPU hardware, so, this creates a time overhead because of the data transference between the cache memory to CPU and GPU. In contrast, the Mustang V100 MX8's architecture is specifically designed for DNN

inference processing with 8 cores of Myriad X processors ensembled at USB 3.2 speed. The processing capabilities of each core is much lower than those of the CPU and GPU, but when several inference instances are executed, it runs more efficiently. In addition, each Myriad X core has a dedicated memory (512MB for each core) to load DNN model weights. Thus, the inference processing is parallelizable at 100 % and there is not data transfer overhead.

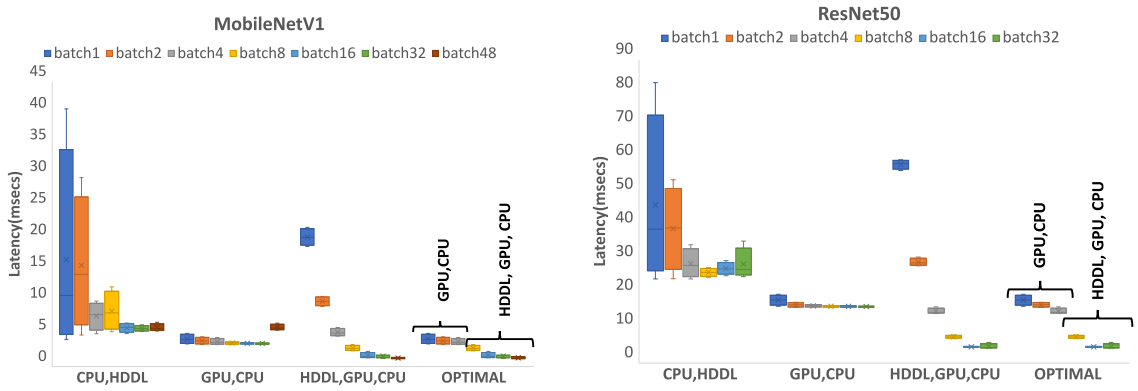


Figure 4.7: Comparison between DNN inference performances obtained by the heterogeneous deployment optimizer vs manual heterogeneous configurations on a TANK AIoT Dev. Kit with a Mustang-V100-MX8 DNN acceleration card.: (a) MobileNetV1; (b) ResNet-50.

Figure 4.8 shows that more stable inference results are obtained in the Jetson, with smaller standard deviation of latency times for all batches and configurations. It also shows that the GPU performs better than the <DLA, GPU> configuration. One of the main reasons is because the capacity of each DLA is 2.5 TOPS, while the GPU can execute 11 TOPS with FP16 precision. Also, even though the DLA's design is based on 4 highly configurable modules (convolution, normalization, activations, data transfer), deploying DNNs to this hardware faces the following drawbacks: First, the current implementation of this module has a limited number of DNN layers and thus those not supported are transferred to the GPU adding an important overhead. Moreover, each DLA core is only able to execute 4 batch streams in parallel. Therefore, even though this NVDLA architecture is innovative, and its low power consumption (0.5-1 Watt) is interesting for IoT platforms, it is still far from the GPU's processing capabilities. In contrast, the NVidia Volta design contains 8 Streaming Multiprocessors (SM) and each SM includes 64 CUDA cores, 8 Tensor Cores. Also, it has ultra-fast memory access with

128KB of L1 cache memory per Volta SM and sharing 512KB L2 offering faster access than previous generations.

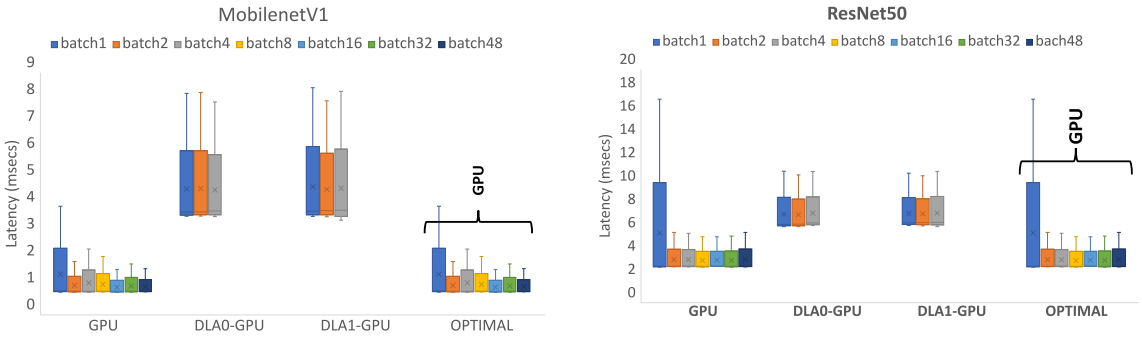


Figure 4.8: Comparison between DNN inference performances obtained by the heterogeneous deployment optimizer vs manual heterogeneous configurations on Jetson Xavier AGX 32GB.; (a) MobileNetV1; (b) ResNet-50

The second experiment consisted of measuring the performance of MobileNetV1 and ResNet-50 for different image batch sizes with different inference configurations, but with SSD-MobileNetV1 being constantly executed as background "computing noise" in different computing processors. Figure 4.9 and Figure 4.10 show the influence of this noise and the configurations that would select our approach in each case.

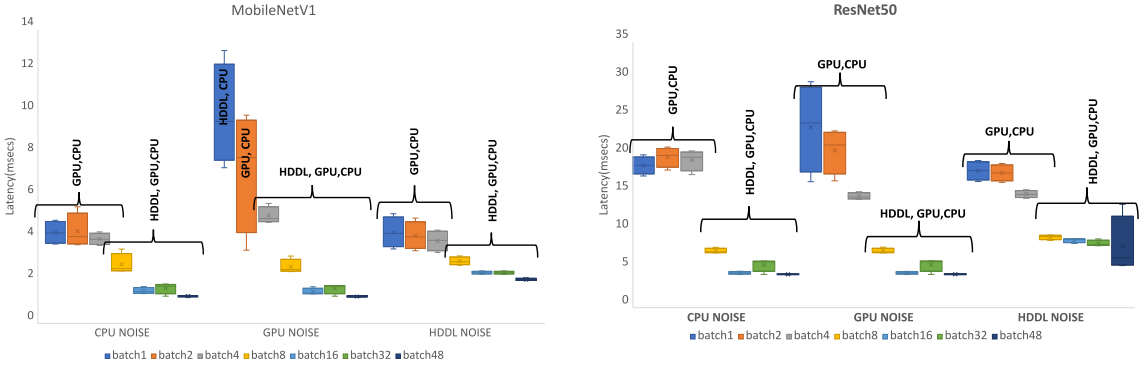


Figure 4.9: The influence of the background hardware usage with heterogeneous deployment optimizer decisions on TANK AIoT Dev. Kit with a Mustang-V100-MX8 DNN acceleration card: (a) MobileNetV1; (b) ResNet-50.

This influence of the background execution is clearly visible in the Figure 4.9. For example, when the CPU is selected for background execution, the <GPU, CPU> and

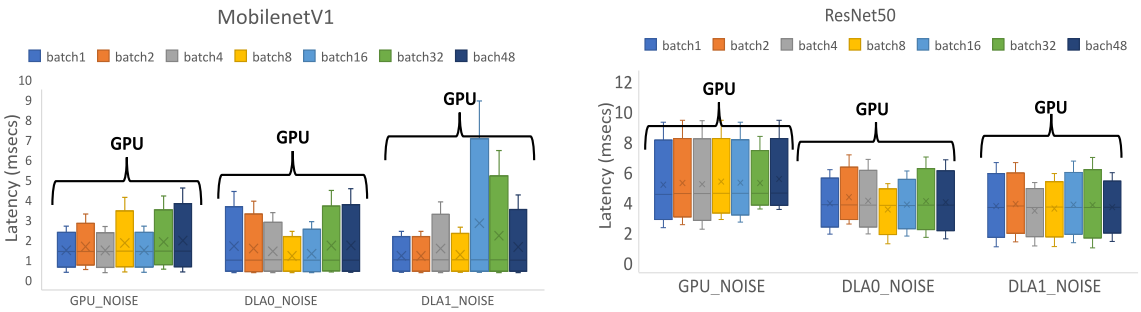


Figure 4.10: The influence of the background hardware usage with heterogeneous deployment optimizer decisions on a NVidia Jetson Xavier AGX: (a) MobileNetV1; (b) ResNet-50

<HDDL, GPU, CPU> devices are selected as the optimal configurations. But, when the GPU is selected for background execution, the HDDL heterogeneous configuration takes almost all batch configurations as optimal (4, 8, 16, 32, 48). Conversely, Figure 4.10, shows very stable results like those shown in Figure 4.8, and the GPU configuration is always the optimal choice. However, as was expected, the higher latency times on both DNN models are clearly visible because of the background execution of SSD-MobileNetV1.

4.4 Practical deployment examples

This section describes practical deployment examples for deploying the face recognition workflow defined in Figure 4.4. These examples are divided into different subsections based on the runtime scenarios mentioned in section 4.2.2. The first subsection describes a scenario for client applications with no DNN inference capabilities. In particular, when a mobile device requests face recognition functionalities from an IoT gateway using the local network. The second subsection represents a runtime scenario with DNN inference capabilities where all face recognition workflow and application logic are executed on board. This practical deployment target is a mobile robot.

.

4.4.1 Deployment runtime scenario: mobile client + IoT gateway

This deployment scenario includes two components, a smartphone with a client application and the IoT gateway to process face recognition workload.

The IoT gateway mission is to offer face recognition services to devices that cannot perform DNN inference processing. This is typically achieved using low-consumption and powerful AI hardware such as GPUs and ASICs. The DNN models and inference engine run on dedicated software within a containerized runtime. These runtimes publish services to client applications via REST APIs and secure communication utilities. Client devices can access these services, but the application logic and interfaces are implemented on the client side. The biometric i-vectors are stored on the IoT gateway.

The smartphone has a client application with a user interface and uses the application logic to request face recognition functionalities to the IoT gateway. Figure 4.11 shows an example of an Android mobile app implementation, including app activities, interfaces, and communication handlers for the REST API.

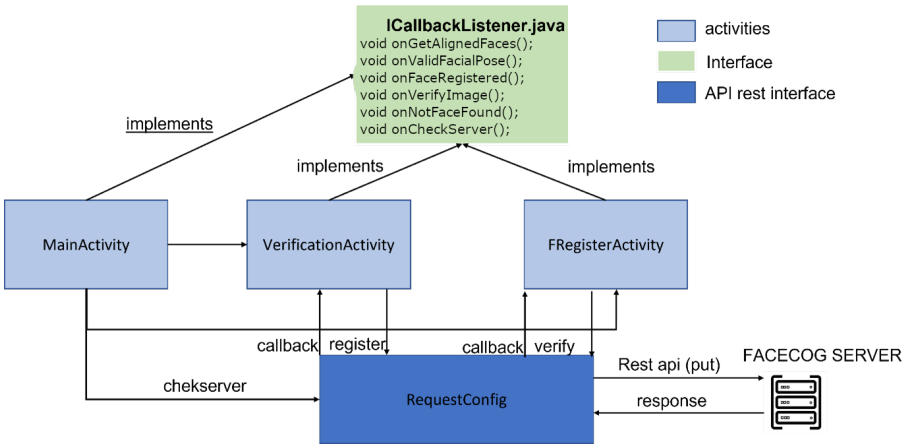


Figure 4.11: The class diagram of the mobile face recognition application (Android app)

The key modules of the application are listed below:

- MainActivity: This is the primary interface of the app and is responsible for configuring the username and server settings (IP and port) for the IoT gateway. It can initiate the user registration or face verification process.
- FRegisterActivity: This module manages the face registration process by invoking services from the IoT gateway.

- VerificationActivity: This module handles tasks related to facial image verification and also invokes services from the IoT gateway.

- RequestConfig: This module manages communication and receives asynchronous callbacks from the IoT rest API responses. The ICallbackListener class implements the necessary function calls based on response results.

To prevent security issues, the biometric i-vectors extracted are stored in the IoT gateway. Figure 4.12 displays images of the various stages of the face recognition workflow along with the graphical user interface.

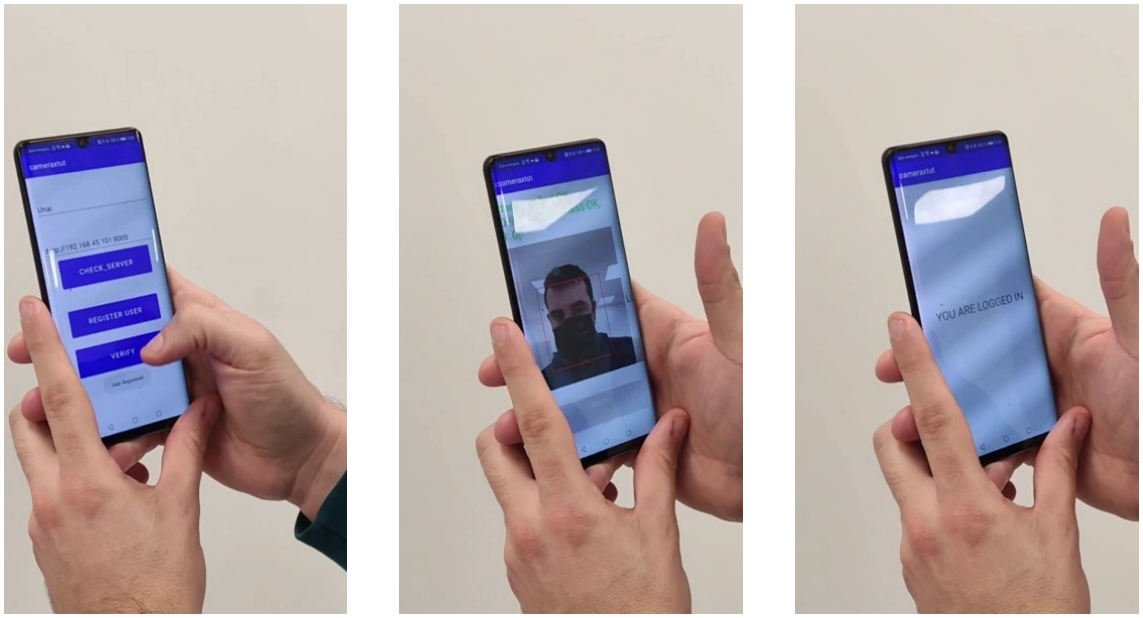


Figure 4.12: Face recognition workflow image examples in mobile scenario.

4.4.2 **Deployment runtime scenario: Mobile robot**

In contrast to the previous deployment scenario, all resources (client application, application logic, and face recognition functionalities) are deployed to PAL Robotics' ARI robot [1].

[1] https://pal-robotics.com/robots/ari/

The robot is equipped with an RGBD camera sensor that works in conjunction with an anti-spoofing DNN model as shown in Figure 4.13. The core functionalities of the face recognition workflow are implemented in the robot operating system (ROS) as part of the face recognition workflow package.

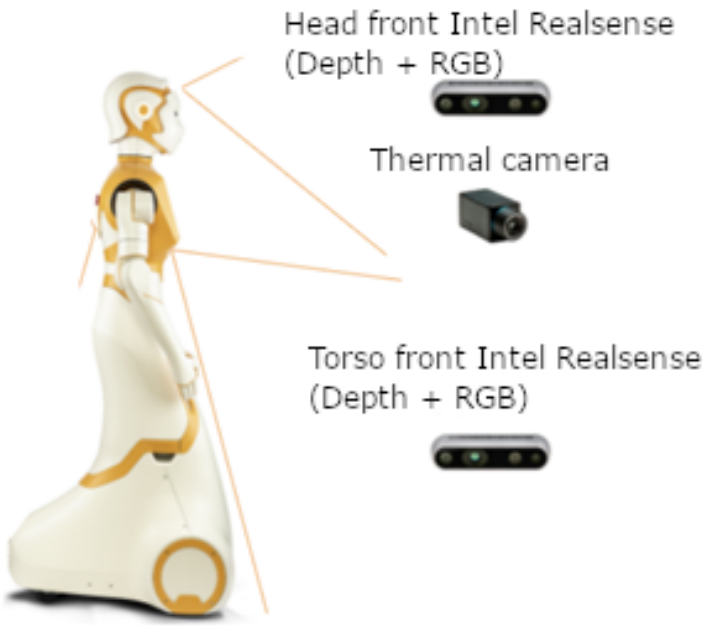


Figure 4.13: PAL Robotics ARI's sensors including RGBD camera sensor.

Face Recognition functionalities rely on the robot's NVidia Jetson TX2 [2] for the execution of the server and client. It also uses ARI's Head RGBD camera (Real Sense) as input. The solution has been integrated through several steps:

- Wrap the Face Recognition server and client as a ROS package.
- Wrap client calls to recognizer services in ROS topics and services. Specifically, the new Python script does the following:
- Subscribe to the ROS topic of the robot's Head RGBD camera (Intel RealSense): /head_front_camera/color/image_raw/compressed. As a ROS topic, the advantage is that multiple digital solutions can use the same camera without overlapping and can be streamed and processed continuously.

[2]https://www.nvidia.com/es-es/autonomous-machines/embedded-systems/jetson-tx2/

- Use of CvBridge python package to convert ROS camera topic to OpenCV image, which is processed by the recognizer.
- Wrap existing recognizer services, by mapping the ROS topics and services.

The robot stores biometric data and shares it with other IoT gateways. A web-based interface is used to access the camera while ROS handles the application logic and communication. Figure 4.14 shows an example image of the graphical user interface in the ARI robot.

Figure 4.14: The front-end interface of the face recognition functionality on the robot's touchscreen.

4.5 Conclusions

In this work, we have presented a knowledge-driven approach for the optimal deployment of DNN-based face recognition solutions in a heterogeneous IoT platform for industrial applications. Our approach tackles its specific challenges in terms of ease of use, hardware heterogeneity, and security. Ease of use is covered by a customized workflow that offers an intuitive interface, and, considers the shared use between the user and companions, by means of automated feedback to automatically take the "selfie"

with the best possible quality. Device heterogeneity is addressed by a smart deployment optimizer, capable of selecting the appropriate DNN inference approach for the targeted device's hardware specifications. Data security is enforced by a scheme that avoids the transmission and the storage of biometric data in the cloud and employs fully homomorphic encryption to perform all face matching operations directly on the encrypted domain. A qualitative comparison with state-of-the-art alternatives and experiments to select the optimal inference approach for different computing situations (different batch sizes and computing noise) reveal its potential for the desired goal. Future work will include further experiments for a greater variety of equipment and a deeper study of the interaction issues with a wider range of users.

CHAPTER

5

Optimizing end-to-end multi-DNN-based video analytics on the edge

This chapter presents an end-to-end method for deploying a multi-DNN-based on-board video analytics system to warn aircraft cabin crew members when passengers store their luggage in areas that could pose a safety risk during critical flight phases such as taxiing, take-off, and landing (TTL). This chapter refers to contribution 5, described in Section 1.3.

The chapter is organized into five sections. Section 5.1 describes the background and challenges of multi-DNN-based onboard video analytics for aircraft cabin readiness verification. Section 5.2 describes the proposed approach to deploy such kind of system. Section 5.3 describes the conducted experimental results to show the potential of this approach. Section 5.4 explains an on-site readjustment procedure of this approach for its practical use in an aircraft cabin. Finally, in Section 5.5, we draw conclusions and suggest ideas for future research.

5.1 Background and challenges

Currently, aircraft cabin operations such as the verification of TTL cabin readiness are done manually. This results in an increased workload for the crew, operational inefficiencies, and a non-negligible risk of human errors in handling safety-related procedures. For TTL, specific cabin readiness requirements apply to the passenger, to the position of seat components and cabin luggage. The usage of cameras and vision-based object-recognition algorithms may offer a promising solution for specific functionalities such as cabin luggage detection. However, building a suitable camera-based smart sensing system for this purpose brings many challenges as it needs to be low weight, with competitive cost and robust recognition capabilities on individual seat level, complying with stringent constraints related to airworthiness certification.

In the aircraft cabin environment, cameras are used as of today for overall cabin monitoring purposes. Current cabin video monitoring systems are characterized by restrained video and image analysis capabilities and are not conceived for specific purposes such as TTL cabin readiness verification. Despite the recent progress in the optimal installation of surveillance cameras to monitor different areas of aircrafts, in practice, the captured images are not being fully exploited. Moreover, different format images and cameras should need to be concealed to exploit the captured images, including AI to help the crew in handling safety procedures. Building a camera-based intelligent system for this purpose reaching the highest Technology Readiness Level (TRL) [125], i.e., TRL9, "an actual system proven in an operational environment", requires overcoming many challenges. With the currently available DNN-based methodologies and equipment this process would start in TRL2, i.e., "a technology concept formulated", and the next step would be to build a TRL3 "experimental proof of concept". This transition from TRL2 to TRL3 is not evident, and relevant technological factors must be analyzed in detail.

Figure 5.1 shows the conceptual design of such a system and the kind of images that would be captured from cameras over the seats and the corridor. In these examples, luggage is placed in different wrong areas for TTL, such as on the corridor's floor and seats, and passengers can also totally or partially occlude the luggage depending on their locations and poses.

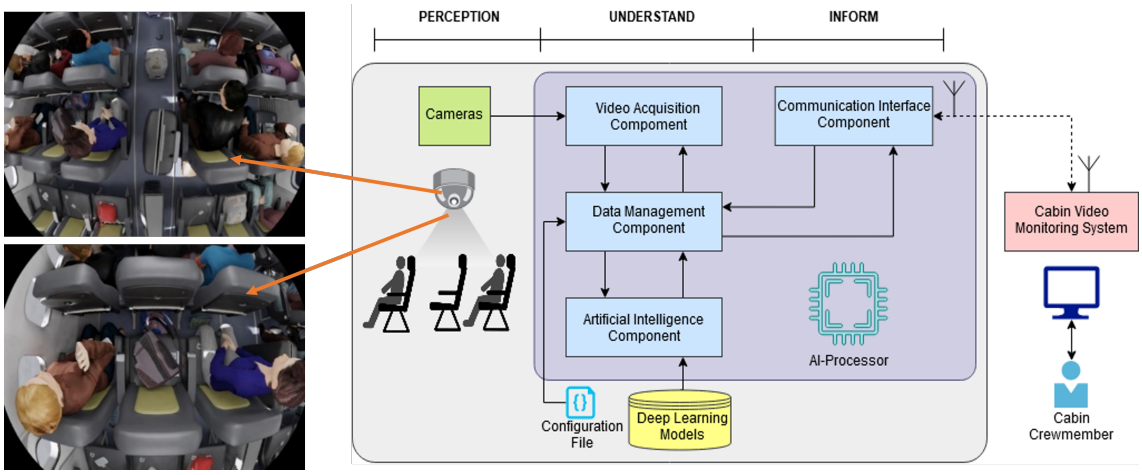


Figure 5.1: Conceptual design of a camera-based intelligent system for digitalized on-demand aircraft cabin readiness verification, and examples of the kind of images that would be captured from cameras installed over the seats and the corridor.

This setup could also be used for other applications such as checking the seat occupancy, whether tray tables are in stowed positions, and identifying dangerous behaviors of passengers, etc.

The economic viability of the system requires minimizing the number of cameras to be installed. This means that each should cover the maximum possible area, e.g., two seat-rows. As it can be observed, the kind of lenses that allow this significantly distort the image content. Thus, the appearances of the visualized objects can be quite different depending on the image region where they are located and the camera position. All these factors are relevant for the design of the computer vision and machine learning algorithms [126]. Captured images would then be processed by AI-processors, which in our context, are edge computing devices that include AI-accelerator(s), i.e., a new generation of CPUs, GPUs, FPGAs, and alternative chips, neural processing units (NPUs), specially designed for the optimal deployment of DNNs [127]

Currently, the most advanced computer vision algorithms rely on DNNs for tasks such as object detection [128], image classification [129], etc. Object detection DNNs can be used to obtain bounding boxes of objects and passengers in an image, as well as distinguish between different kinds of object classes. On the other hand, image classification DNNs can be applied to image regions containing one seat or aisle section to label the image content with learned concepts (Figure 5.2).

Figure 5.2: Examples of image crops to be processed by multiple DNNs to verify the correct positioning of the luggage for the cabin readiness verification.

This can help determine whether luggage positioning in that region is correct or not. The advantage of using image classification DNNs in this context is that training data can be labeled more easily, and that DNNs for classification are also much more efficient.

To obtain accurate classification results, samples from the same class should have similar visual appearances and be significantly different from the other class. However, as shown in Figure 5.2, this is not the case in our problem as there could be many different kinds of objects and people involved, with widely varied appearances and spatial relations. In addition, there are objects whose presence is not a problem for TTL cabin readiness, such as magazines, books, food, smartphones, tablets, jackets, wallets, etc., which we will refer to as “non-cabin luggage”. On the contrary, “cabin luggage” would include backpacks, bottles, briefcases, camcorders, hats, laptops, shopping bags, suitcases, tote bags, etc.

Thus, considering this kind of variability, we can divide the two classes into fine-grained subclasses grouped according to different visual appearances. Figure 5.3 shows an example of this subdivision for seats in standard seat rows of the aircraft, represented as a three-level hierarchy of classes. The top level contains the two-goal classes (correct or incorrect).

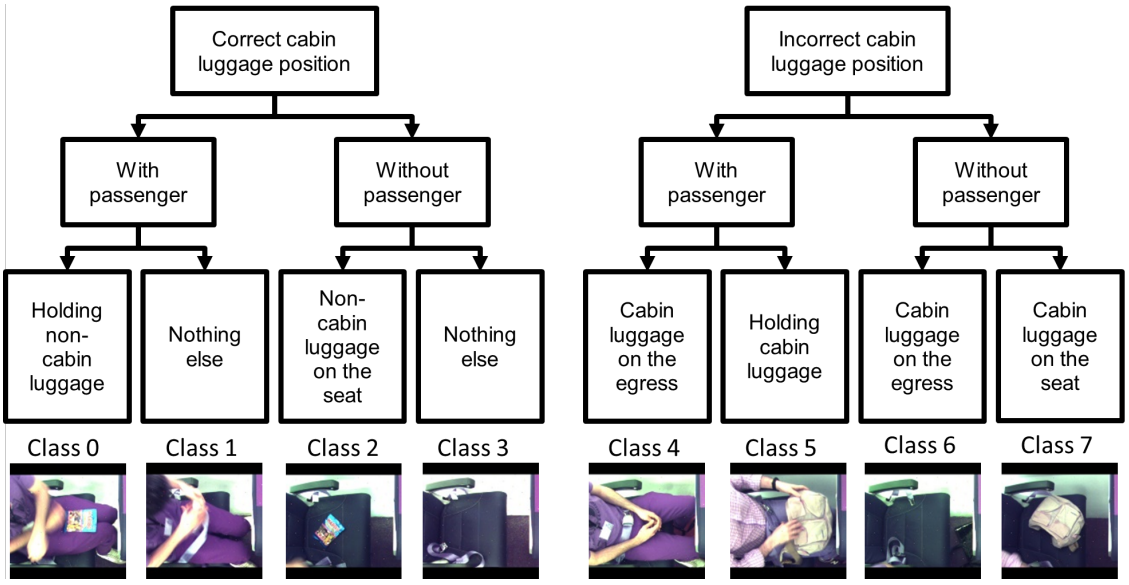


Figure 5.3: Example of subdivision in the positioning of luggage for TTL cabin readiness as a three-level hierarchy of classes.

The middle level considers the presence or absence of a passenger on the seat, and the lower level considers the absence or presence of objects, their type, and their positioning.

Going through the hierarchy from the top level to each class of the lower level, we obtain more specific and richer descriptions of the correct and incorrect situations. A classification DNN could be trained by taking these lower-level subclasses independently. However, a better approach is to leverage this extra information to improve the accuracy and reliability of the system through learning paradigms such as multi-task learning [130, 131] and metric-guided prototype learning [132]. In this chapter, we analyze how this could be done in our context.

Another problem to be tackled in this system is that as the appearances of seats vary depending on their placement in the image—due to perspective and the image distortion— multiple DNNs would needed to analyze the whole scene. Each DNN would learn the specific kind of appearances expected in each image region. For that, how datasets are designed and used for training is another key factor to be considered, especially in this kind of context with visual specificities not normally present in generalist datasets, like ImageNet [60], COCO [105] or OpenImages [133]. For this purpose, we also present in this chapter the SmaCS dataset [134] to facilitate future research. Considering

that in practice, data could usually come from different data sources (i.e., various cabins, real or synthetic), we consider domain adaptation techniques to enhance generalization performance [135].

Deploying a multi-DNN-based multi-camera system on an onboard embedded processor for aircraft cabin readiness verification, consuming minimal power, poses significant challenges in achieving accuracy, robustness, and responsiveness. In the following sections, we address this challenging problem by making the following contributions:

1. An approach to deploy on-board video analytics for in-flight TTL cabin readiness verification with an optimal trade-off between response delay and power consumption.

2. Metric-guided multi-task domain-adversarial prototypical networks (MMDAPNs) for efficient and robust image classification, exploiting the hierarchical structure of classes.

3. An optimal multi-MMDAPN processing pipeline tailored to the embedded processor's heterogeneous computing capabilities.

4. Experimental results with the SmaCS dataset [134], comprising around 30K images captured in a cabin mockup and 7K generated with computer graphics depicting various situations involving passengers and objects.

5.2 Methodology

5.2.0.1 MMDAPN architecture design and training

DNNs rely on data to learn and make predictions. The accuracy and reliability of DNNs depend on how the data is handled during training, such as identifying commonalities between classes and the relationships between labels. Multi-task learning [130] aims to improve the learning performance of multiple related tasks by sharing valuable information between them. In our case, the primary task is to classify the top-level classes, while auxiliary tasks involve classifying the subclasses. However, identifying commonalities between different subclasses could make it challenging to distinguish between

the main classes. Metric-guided prototype learning [132] can help address this issue by making DNNs focus on relevant image features to distinguish between classes based on semantic hierarchical priors.

Designing a system that uses DNNs requires careful consideration of the amount and quality of data available to train the model. To ensure better generalization, training datasets should be designed with care, gathering as many samples as possible of each subclass from different real cabins to avoid bias among classes. Synthetic data can help with this process [136] [137], but it isimportant to handle the domain gap between different data sources (i.e., various cabins, real or synthetic). Domain adaptation methods [135] can help DNNs extract robust cross-domain features.

Our proposed MMDAPN architecture improves the classification accuracy and reliability of DNNs by combining multi-task, metric-guided prototype, and domain adversarial learning paradigms. Figure 5.4 shows the architecture of MMDAPNs used to train classification models that exploit the semantic information present in the hierarchy of classes for aircraft cabin readiness verification with data from different sources (e.g., synthetic and real cabins). The deployment of multiple MMDAPNs to analyze images like those shown in Figure 5.2 enables efficient on-board video analytics of in-flight TTL cabin readiness verification.

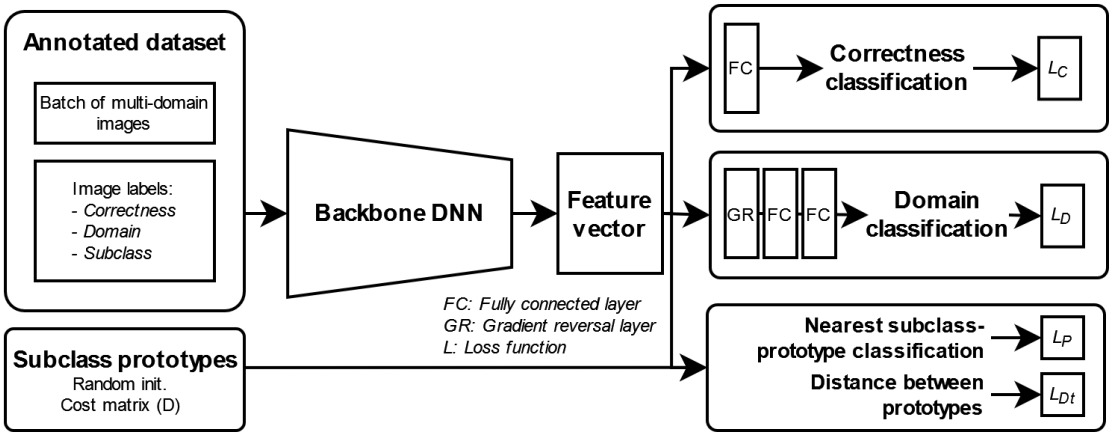


Figure 5.4: The architecture of metric-guided multi-task domain adversarial prototypical networks (MMDAPNs).

In our context, we construct MMDAPNs around a backbone classification DNN. Given the limited computational resources of the onboard embedded processor, we

consider efficient DNNs such as EfficientNetV2 [64]. During training, the MMDAPN architecture aims to optimize three tasks using multi-task learning: (1) classifying the main top-level classes (i.e., correct or incorrect luggage positioning), (2) classifying the source domain (e.g., cabin 1, cabin 2, cabin 3, etc.), and (3) classifying the fine-grained scene descriptive subclasses (Figure 5.3).

The input data comes from two sources: (1) an annotated dataset of images gathered from various cabins (real or synthetic), and (2) subclass prototypes generated by the user-defined semantic hierarchical priors that guide the learning process. The former is introduced batch by batch in each iteration of the training process, equally distributing the data from each data source (i.e., cabin 1, cabin 2, etc.). Data samples include labels for three tasks: luggage positioning correctness (0 or 1), domain ID (integer value), and scene descriptive subclass ID (integer value). The latter uses a finite metric on the hierarchical class set to supervise a prototypical network that associates each class with a representation or prototype and classifies observations according to the nearest prototype [132]. More specifically, the metric is defined by semantic distances among subclasses under the form of a cost matrix that acts as a distortion-based regularizer for the prototypical network.

Eq. (5.1) shows the loss function of MMPADNs. $\mathscr{L}_C$ is a binary cross-entropy loss for the luggage positioning correctness classification, $\mathscr{L}_D$ is a binary or softmax cross-entropy loss for the domain classification (depending on whether there are two or more data source domains), $\mathscr{L}_P$ is the loss for the subclass-prototype classification and $\mathscr{L}_{Dt}$ the loss for the metric-distortion-based regularization. The loss weights λ_C, λ_D, λ_P and λ_{Dt} are floating-point values between 0 and 1, empirically selected to balance the contribution of each loss.

$$\mathscr{L} = \lambda_C \mathscr{L}_C + \lambda_D \mathscr{L}_D + \lambda_P \mathscr{L}_P + \lambda_{Dt} \mathscr{L}_{Dt} \tag{5.1}$$

By including $\mathscr{L}_D$ in the loss function, the MMPADN learns how to distinguish each domain during training and adjusts the backbone DNN's weights to maximize the outcome of the rest of the learning tasks in all domains. This helps the DNN learn the most robust features for all domains simultaneously.

We rely on the approach of metric-guided prototypes proposed in [132] for the definitions of $\mathscr{L}_P$ and $\mathscr{L}_{Dt}$. We consider a training dataset $\mathscr{N}$ of N elements $x \in \mathscr{X}^N$ with

ground truth classes $z \in \mathcal{K}^N$, hierarchically organized. A user-defined cost matrix D defines a finite metric that considers the shortest path between nodes of the hierarchical $\mathcal{K}$ classes. Ω is the embedding space that forms a continuous metric space when equipped with the distance function $d : \Omega \times \Omega \mapsto \mathbb{R}_+$. Eq. (5.2) shows an example of D for the class hierarchy example of Figure 5.3, where rows and columns follow the same order as the lower-level subclasses, i.e., 0: *Correct_with_passenger_holding_non-cabin_luggage*, 1: *Correct_with_passenger_nothing_else*, 2: *Correct_no_passenger_non-cabin_luggage_on_the_seat*, etc.

$$D = \begin{bmatrix} 0 & 2 & 4 & 4 & 6 & 6 & 6 & 6 \\ 2 & 0 & 2 & 4 & 6 & 6 & 6 & 6 \\ 4 & 2 & 0 & 2 & 6 & 6 & 6 & 6 \\ 4 & 4 & 2 & 0 & 6 & 6 & 6 & 6 \\ 6 & 6 & 6 & 6 & 0 & 2 & 4 & 4 \\ 6 & 6 & 6 & 6 & 2 & 0 & 2 & 4 \\ 6 & 6 & 6 & 6 & 4 & 2 & 0 & 2 \\ 6 & 6 & 6 & 6 & 4 & 4 & 2 & 0 \end{bmatrix} \tag{5.2}$$

In this example, we distinguish four semantic distance values: 0 for the lower-level subclass with respect to itself, 2 for lower-level subclasses of the same top-level and mid-level groups, 4 for subclasses of the same top-level group and different mid-level class, and 6 for subclasses of different top-level groups.

Following this nomenclature, a prototypical network is characterized by an embedding function $f : \mathcal{X} \rightarrow \Omega$ and a set $\pi \in \Omega^{\mathcal{K}}$ of K prototypes. Based on [138], a prototypical network associates with an observation x_n the posterior probability over its class z_n, as shown in Eq. (5.3).

$$p(z_n = k|x_n) = \frac{\exp(-d(f(x_n), \pi_k))}{\sum\limits_{l \in \mathcal{K}} \exp(-d(f(x_n), \pi_l))}, \forall k \in \mathcal{K} \tag{5.3}$$

Thus, $\mathcal{L}_P$ is defined as the normalized negative log-likelihood of the true classes, as shown in Eq. (5.4). This loss encourages the embedding function $f(x_n)$ to be close to the prototype π_{z_n} and far from the other prototypes.

$$\mathcal{L}_{\mathcal{P}}(f, \pi) = \frac{1}{N} \sum_{n \in \mathcal{N}} \left(d(f(x_n), \pi_{z_n}) + \log \left(\sum_{l \in \mathcal{K}} \exp(-d(f(x_n), \pi_l)) \right) \right) \tag{5.4}$$

Finally, $\mathscr{L}_{D_t}$ is defined as shown in Eq. (5.5). This loss enforces a metric-consistent prototype arrangement to avoid conflicting with the second term of $\mathscr{L}_P$ by scaling prototype coordinates in Ω by a scalar factor s. Thus, it encourages a low distortion between $s \cdot \pi$ scaled prototypes and D.

$$\mathscr{L}_{Dt}(\pi) = \frac{1}{K(K-1)} \min_{s \in \mathbb{R}_+} \sum_{k,l \in \mathscr{K}^2, k \neq l} \left(\frac{sd(\pi_k, \pi_l) - D[k,l]}{D[k,l]} \right)^2 \tag{5.5}$$

5.2.1 Optimal multi-MMDAPN processing pipeline

The installation's characteristics are determined by the cabin areas to be covered and the minimum cabin luggage size. These characteristics include the number of cameras, their placement, their lens characteristics, the number of image regions of interest (ROIs) analyzed (Figure 5.2), and their resolution. However, deploying the multi-MMDAPN-based approach optimally in a heterogeneous embedded processor onboard that satisfies the onboard response latency and power consumption constraints is not straightforward.

The proposed end-to-end processing pipeline for this purpose is shown in Figure 5.5 and is composed of four modules: (1) multithreaded image capture, (2) batched image pre-processing, (3) multi-MMPADN inference, and (4) response post-processing.

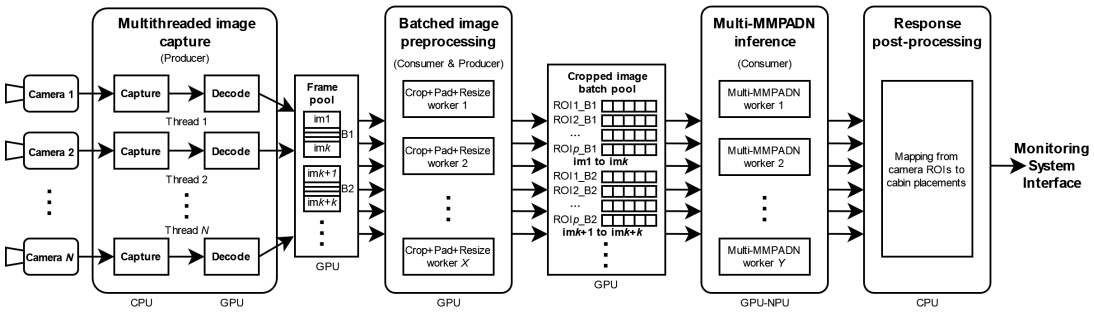


Figure 5.5: Multi-camera multi-MMDAPN-based end-to-end processing pipeline.

The first module produces a full-image frame pool in the GPU by capturing images from n cameras scheduled by n CPU threads and decoding them in the GPU. The second module, composed of x pre-processing workers, consumes x batches of k images from this frame pool to produce a pool of batched cropped images corresponding to p ROIs that are padded and resized to match the MMPADN's input resolution in

GPU. Then, the third module, composed of y multi-MMPADN inference workers in GPU-NPU, consumes y multi-ROI batches from this pool and delivers the classification results to the fourth module that maps them in CPU to cabin placements to be shown in the monitoring system interface.

Parameter	Description
k	number of images in a batch
p	number of ROIs
x	pre-processing threads / number of batches
y	number of DNN inference threads

Table 5.1: Multi-MMDAPN processing pipeline parameter description

The values of k, p, x, and y must be carefully selected, as well as the XPU (typically, GPU and/or NPU) where the MMPADN models will be deployed for inference to obtain the optimal trade-off between latency and power consumption for a targeted embedded processor that complies with onboard constraints. To help achieve this goal, we propose algorithm 2.

Algorithm 2 Multi-MMDAPN processing pipeline optimization for the multi-camera onboard system.

procedure PIPEOPT($\mathbf{m}, \mathbf{I}, \mathbf{c}, k_{\max}, x_{\max}, y_{\max}, P_{thr}, \Delta_{thr}$)
  c_{opt} = None
  $k_{\text{opt}} = x_{\text{opt}} = y_{\text{opt}} = -1$
  $S_{\min}$ = BigNumber
  for c in **c do**
    for $k \leftarrow 1$ to $k_{\max}$ **do**
      for $x \leftarrow 1$ to $x_{\max}$ **do**
        for $y \leftarrow 1$ to $y_{\max}$ **do**
          $P_{\max}, \Delta_{\text{avg}} \leftarrow \text{eval}(\mathbf{m}, \mathbf{I}, c, k, x, y)$
          if $P_{\max} < P_{\text{thr}} \& \Delta_{\text{avg}} < \Delta_{\text{thr}}$ **then**
            $S \leftarrow d_{\min}((P_{\max}, \Delta_{\text{avg}})$
            if $S < S_{\min}$ **then**
              $c_{opt} = c$
              $k_{opt} = k$
              $x_{opt} = x$
              $y_{opt} = y$
  return $c_{\text{opt}}, k_{\text{opt}}, x_{\text{opt}}, y_{\text{opt}}$

Algorithm 2 evaluates the performance of various MMPADN-XPU deployment configurations **c** using a testing dataset **I** of n videos processed by **m** MMDAPNs. The algorithm measures the average latency Δ_{avg} and maximum power consumption P_{max} for each deployment candidate configuration. It then selects all the configurations that meet the maximum acceptable P_{thr} and Δ_{thr} values. The optimal deployment configuration is selected based on the minimum Euclidean distance S_{min} between the normalized latency and power values and the origin (0,0) on the latency-power plane.

5.3 Experiments and discussion

5.3.1 The SmaCS dataset

To collect the required data for the SmaCS dataset [134], we built a cabin mockup and generated synthetic data with 3D graphics. The mockup consists of one side of three cabin seat rows, the aisle, and the covering cabin exterior made of polystyrene (Figure 5.6). It is illuminated by three possible light sources: (1) natural light coming from the room's windows beside it, (2) artificial light on top of the room, and (3) a spotlight beside the cabin's window to emulate directional sunlight. We recorded the data on different days, varying the illumination conditions. To monitor two seat rows, we put two cameras on top of the four seats closer to the window and another on top of the aisle to monitor the aisle and the two seats beside it.

We created a recording protocol that 18 participants followed to simulate different situations with cabin and non-cabin luggage objects (Figure 5.7). The recording protocol specified the acting guide for passengers, such as how long they should sit in a specific seat and where they should place the carried object, if any. The goal of the recording protocol was to recreate a balanced number of situations, such as correctly and incorrectly placed luggage.

On the other hand, we generated the synthetic data by following the methodology explained in [136]. We modeled 22 different object types to simulate typical cabin and non-cabin luggage objects, a cabin model representing a Boeing 737 aircraft, and a set of human models with different poses and appearances for the seated passengers. The appearance of the cabin and objects is randomized for higher variability.

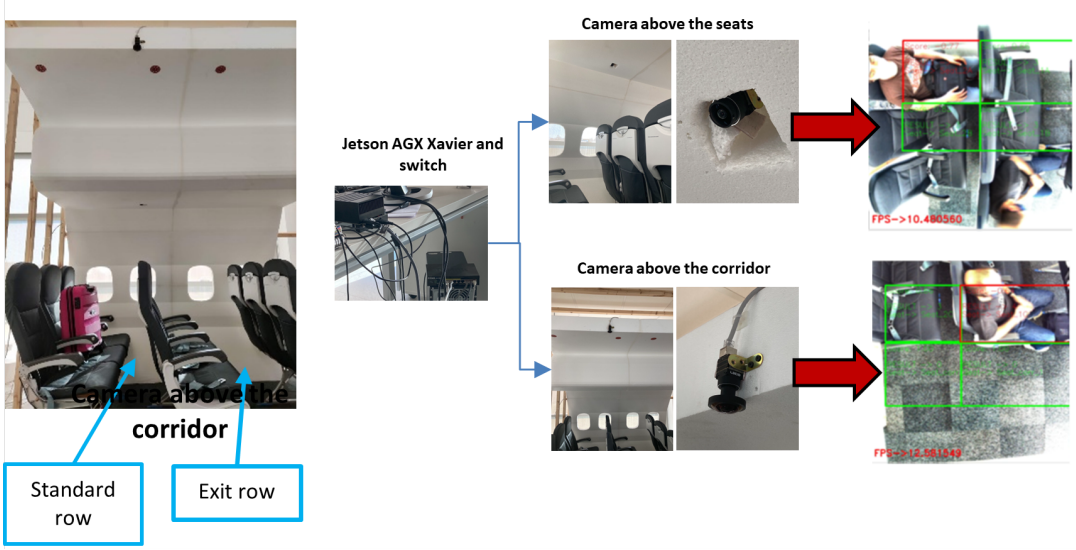


Figure 5.6: The cabin mockup, AI-processor and camera setup used for the experiments.

Figure 5.7: The objects used for the MMDAPN experiments.

Figure 5.8 shows some images of the virtual environment that were used to generate synthetic data.

In [134], we published the data of one of the seats (the closest to the cabin's window of the back row). For this seat, we obtained approximately 30K real and 7K synthetic samples. Figure 5.9 shows some examples of both real and synthetic ROI images. Figure 5.10 shows the distribution of the collected data regarding the defined subclasses.

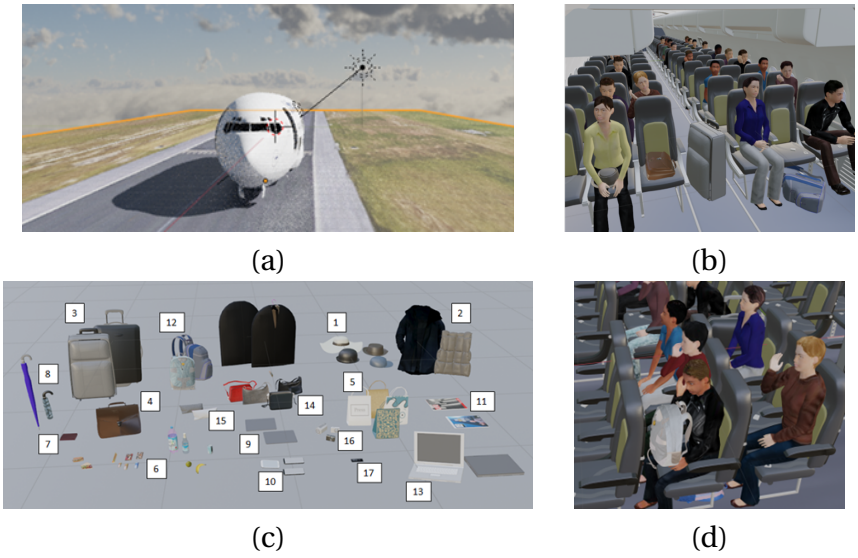


Figure 5.8: Examples of the virtual environment used to generate synthetic images. (a) Exterior view of a Boeing 737 airplane. (b) Interior perspective view of the airplane cabin. (c) Cabin and non-cabin luggage virtual objects. (d) View of passengers holding their luggage.

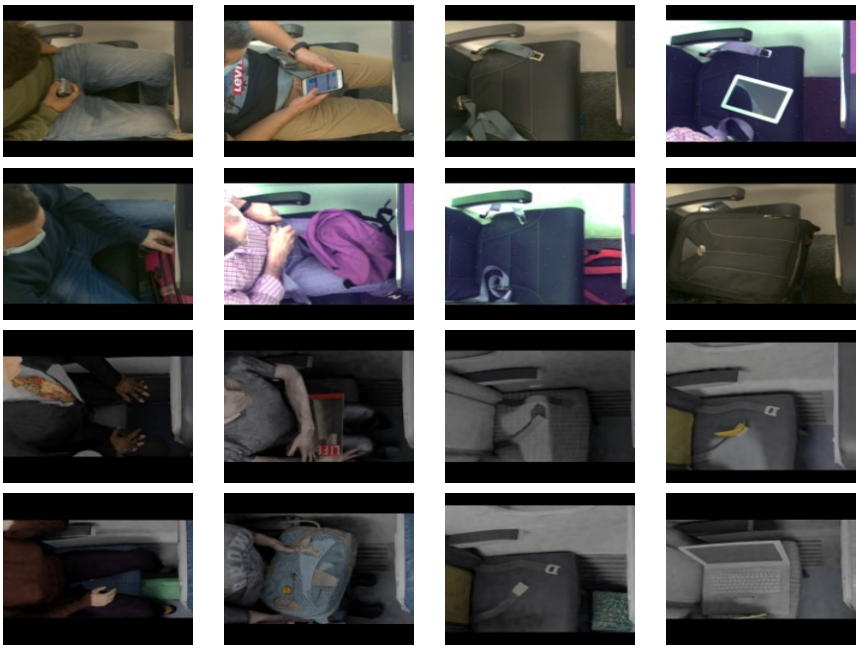


Figure 5.9: Examples of image ROIs from captured images and synthetic images for the different fine-grained subclasses. The first two rows refers to real images while the third and fourth rows refers to synthetic

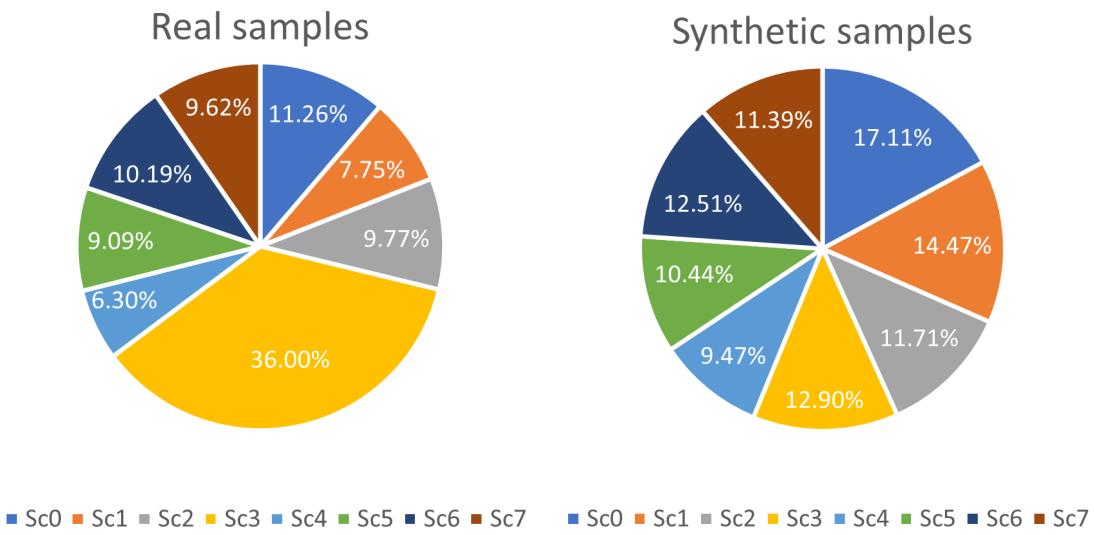


Figure 5.10: Distribution of the eight fine-grained subclasses in the captured and generated samples. Subclasses with index from 0 to 3 represent correct situations and from 4 to 7 incorrect ones (cabin luggage incorrectly placed).

It can be seen that the number of samples per subclass in the synthetic data is balanced but not in the real data. The reason is that even if the actors followed a recording protocol, capturing samples representing the "empty seat" subclass (subclass 3) in the intermediate moments between two situations is inevitable.

5.3.2 Accuracy and reliability evaluation

To test the accuracy and reliability of the proposed MMDAPN, we separated a subset of approximately 2,700 images for testing (real samples) and used the rest of them (real and synthetic) to train the model. The testing images do not contain visually similar situations (e.g., the same person with the same object and illumination) as it haappens in the training data.

We used EfficientNetV2-B0 [64] as the backbone network of the MMDAPN with pre-trained weights from the ImageNet dataset [60]. We resized input images to 300×300 pixels, as it is the minimum size that allows visualizing small objects for classification, and empirically set the values of $\lambda_C = 0.8$, $\lambda_D = 0.6$, $\lambda_P = 1$, $\lambda_{Dt} = 1$. We used D as shown in Eq. 5.2.

Table 5.2 presents a comparison of MMDAPN's performance on the test set against nine state-of-the-art alternatives. These alternatives include: (1) an image classifier based on EfficientNetV2-B0 [64], trained solely on overall correctness classes, (2) DANN with EfficientNetV2-B0 as the backbone [137], trained in a similar manner, (3) EfficientNetV2-B0 trained using subclasses and inferring overall correctness from them, (4) DANN with subclasses, (5) EfficientNetV2-B0 with multi-tasking, where the main task is overall correctness and subclasses serve as auxiliary tasks [131], (6) DANN with multi-tasking, (7) EfficientNetV2-B0 with prototypes of subclasses [132], used for inferring overall correctness, and (8) DANN with prototypes, following the same approach as (7), and (9) EfficientNetV2-B0 with multi-tasking and prototypes of subclasses.

Table 5.2: Comparison of MMDAPN with subclass accuracy and overall correctness on the test set with state-of-the-art alternatives (Sc: subclass).

	Correct cabin luggage position				**Incorrect cabin luggage position**				
Method	**Sc0**	**Sc1**	**Sc2**	**Sc3**	**Sc4**	**Sc5**	**Sc6**	**Sc7**	**Correctness**
EfficientNetV2 [64]	-	-	-	-	-	-	-	-	85.29%
EfficientNetV2[64] subclasses	76.09%	**100%**	94.93%	**100%**	73.19%	66.67%	**100%**	72.46%	89.02%
EfficientNetV2 [64] + multi-task [131]	77%	96%	**100%**	**100%**	74%	65%	**100%**	**100%**	92%
EfficientNetV2 [64] + prototypes[132]	68%	97%	54%	**100%**	45%	66%	**100%**	**100%**	88%
EfficientNetV2 [64] + multi-task [131] + prototypes[132]	76%	99%	96%	**100%**	57%	66%	**100%**	**100%**	90%
DANN [137]	-	-	-	-	-	-	-	-	94.4%
DANN [137] subclasses	**79%**	94%	96%	**100%**	**77%**	70%	**100%**	99%	91%
DANN [137] + multi-task [131]	60.7%	99.7%	97.9%	**100%**	76.3%	**100%**	**100%**	99.3%	95.0%
DANN [137] + prototypes [132]	62%	**100%**	99%	**100%**	54%	79%	**100%**	99%	90%
MMDAPN (ours)	65.4%	**100%**	**100%**	**100%**	75.2%	97.7%	99.0%	**100%**	**96.9%**

The results reveal that all methods achieve perfect detection accuracy for an empty seat (subclass 3), with a 100% accuracy rate across all cases. However, the most challenging scenarios arise in subclass 0 (passenger holding non-cabin luggage), reaching a maximum accuracy of 79%, and subclass 4 (cabin luggage on the egress with a passenger), reaching a maximum accuracy of 77%. This outcome is to be expected since an empty seat provides visual consistency, whereas the presence of a passenger introduces increased visual variability. Additionally, distinguishing between non-cabin and cabin luggage can be difficult, particularly when the luggage is partially occluded by passengers. Similarly, detecting luggage on the egress can present challenges for the same reason.

Among all the methods considered, MMDAPN achieves the highest accuracy in overall correctness classification, achieving a remarkable accuracy of 96.9%. This result surpasses the second-best method (DANN + multi-task) by 1.9% and outperforms

the least accurate method (EfficientNetV2) by a substantial margin of 11.61%. Furthermore, MMDAPN also achieves the highest scores in four out of the subclasses, which is the highest number of top rankings among all the methods. However, it is worth emphasizing that the most significant outcome in this particular use case is the overall correctness classification.

The closest competitor to MMDAPN is DANN with multi-tasking, but MMDAPN consistently yields superior results across all categories except for three subclasses (subclasses 4, 5, and 6), where multi-task DANN achieves marginally better performance.

The results further demonstrate that incorporating both the multi-task approach, which integrates subclass information, and the matrix-guided prototypical component, which influences the positioning of subclasses in the feature space, leads to improved classification accuracy. Based on these findings, we conclude that the MMDAPN architecture effectively classifies whether cabin luggage is correctly or incorrectly placed on the monitored seat.

To observe the impact of the metric-guided prototypical network component on the arrangement of features in the feature space, we extracted the features from the training images and employed principal component analysis (PCA) for dimensionality reduction, aiming to visualize the results. In Figure 5.11, we present the extracted features using MMDAPN with and without the metric-guided prototypical network component.

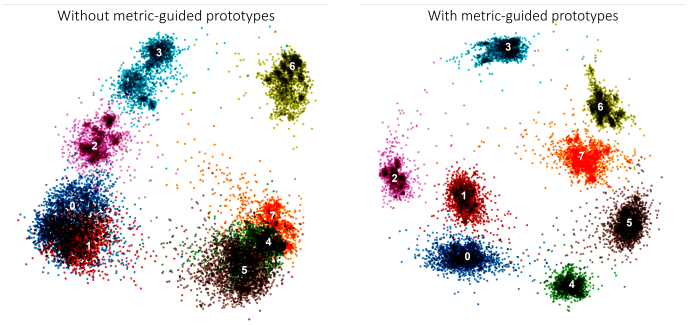


Figure 5.11: Image features visualization after PCA. Each subclass is represented using a different color. Features are extracted using the trained MMDAPN (right) and the model without the metric-guided prototypical network component (left).

In the case of training without the prototypical metric guidance (Figure 5.11, left), the features of different subclasses appear more entangled in the space, as exemplified

by subclasses 4, 5, and 7. Conversely, when the metric-guided prototypical network component is incorporated (Figure 5.11, right), the features of each subclass exhibit clearer separation, and subclasses that are more similar to each other, such as subclass 6 and 7, are situated closer in the feature space. Consequently, in situations where misclassification occurs, the outcomes would be more meaningful, and the system would be more reliable.

5.3.3 Optimal processing pipeline evaluation

We conducted both qualitative and quantitative evaluations to assess the effectiveness of our optimal processing pipeline. The qualitative evaluation involved comparing various features of our approach with state-of-the-art alternatives. On the other hand, for the quantitative evaluation, we presented experimental results of deploying multi-MMDAPN using a practical example on an NVIDIA Jetson AGX Xavier platform. In Table 5.3, we summarize the qualitative comparison across five different categories.

Table 5.3: Qualitative comparison of our Multi-MMDAPN processing pipeline with respect to alternative state-of-the-art approaches.

Method	DNN types	DNN workload type	Optimization criteria	XPUs involved	Deployment config
[83]	Classifiers	Variable	Δ_{min}	CPU-GPU	Manual
[84]	Classifiers	Variable	Δ_{min}	CPU-GPU	Manual
[85][86]	Detectors	Constant	Max throughput	GPU-NPU	Manual
[87]	Classifiers	Variable	Δ_{min}	GPU	Manual
[88]	Detectors and classifiers	Variable	Δ and RAM trade-off	CPU	Manual
[89]	Classifiers	Variable	Δ_{min}	GPU	Manual
Ours	Classifiers	Constant	Δ and P trade-off	CPU-GPU -NPU	Automatic

As described previously, our approach focuses on constantly operating classification DNNs. While other works may employ detection DNNs or a combination of both types with variable workloads, our multi-DNN deployment aims to handle different situations specific to our use case. Unlike deployment methodologies that primarily prioritize inference speed and memory management, our deployment strategy also takes into account power consumption, optimizing the processing pipeline across the CPU-GPU-NPU subsystems. As illustrated in Figure 5.5, the CPU is responsible for

post-processing and capture tasks, while the preprocessing and multi-DNN inference tasks are offloaded to GPU and NPU hardware accelerators. Considering that the embedded device installed in the aircraft must adhere to low-power specifications, our approach strikes a balance between latency and power consumption measurements. Additionally, in contrast to state-of-the-art alternatives, which require manual configuration of deployment parameters, our approach automatically calculates the optimal configuration parameters for the processing pipeline.

In the quantitative evaluation, we selected JEDI [86] from the methods listed in Table 5.3 as it shares the same DNN workload type as our approach. However, it is important to note that while JEDI uses DNNs for detection (specifically, YOLOv4 [139]), our approach focuses on classification. This distinction implies that JEDI may require more computational resources compared to our pipeline. Furthermore, both our pipeline and JEDI employ different configurations that influence the trade-off between latency and power consumption. As a result, it becomes necessary to evaluate which approach yields the best results when considering their respective configurations.

In order to compare different deployment configurations, we conducted measurements of latency and power consumption using algorithm 2 with the following configuration parameters: $k_{\max} = 15$, $x_{\max} = 15$, $y_{\max} = 6$, $P_{\text{thr}} = 30$W, and $\Delta_{\text{thr}} = 40$ms. We considered various batch sizes, specifically $k = 1,3,5,15$, $x = 1,3,5,15$, and $y = 1,2,3,6$. To calculate the average latency per image Δ_{avg}, we used the formula $(T_l - T_f)/\text{num\_processed\_frames}$, where T_l represents the total processing time and T_f denotes the time spent on the framework. Power consumption measurements were taken every second to prevent system saturation. For the evaluation of JEDI, we used their publicly available implementation [140]. The methodology employed to measure latency and power consumption was consistent for both our approach and JEDI.

The deployment software used for the NVIDIA Jetson Xavier AGX was JetPack 4.6.2, which incorporated TensorRT 8.2.1 and CUDA 10.2. The system was configured for MAXN power mode, and the Jetson clocks were activated. We employed 15 full-HD cameras as input sources for our approach. Specifically, our approach used six ROIs, each trained with an MMDAPN model. This resulted in a total of six MMDAPNs, with EfficientNetV2-B0 [64] serving as the backbone. The cropped images were resized to dimensions of 300×300 pixels. For the purpose of comparison with JEDI, we selected

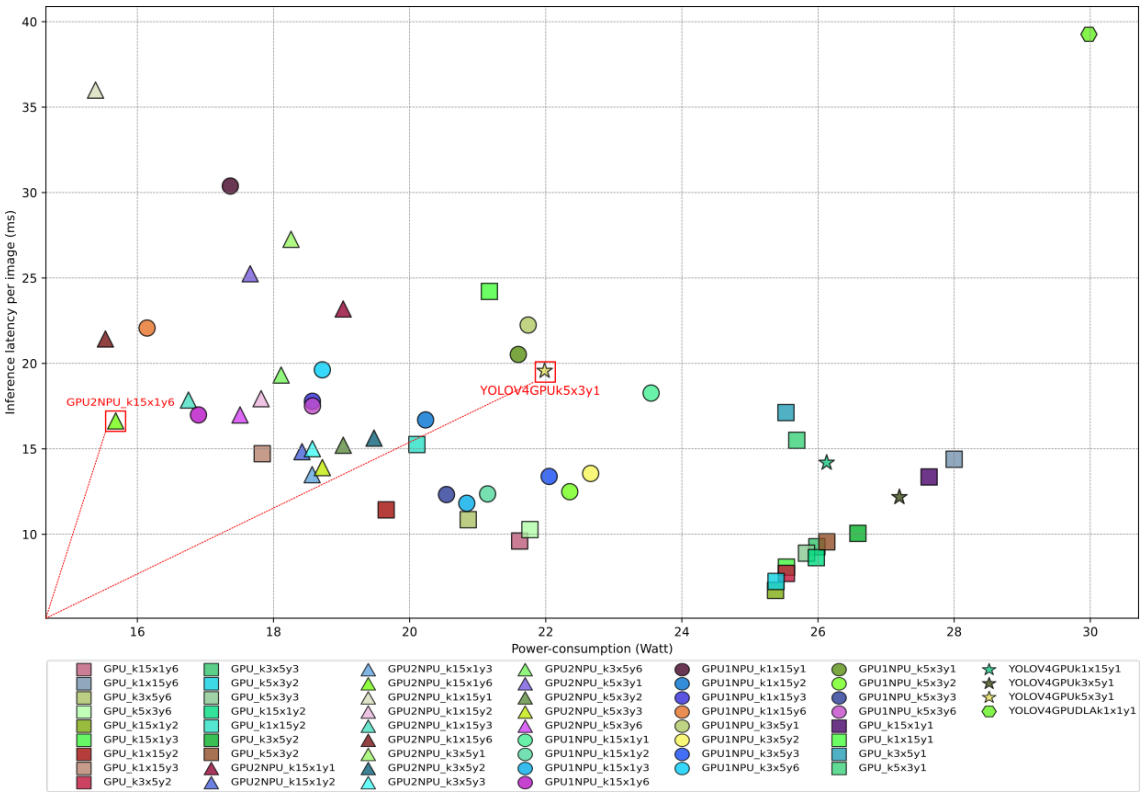


Figure 5.12: The Multi-MMDAPN processing pipeline evaluation to find the optimal deployment configuration with Jetson Xavier AGX 32GB. The optimal configurations are selected in red rectangle.

their optimized implementation of YOLOv4 [139] with an input size of 416 × 416 pixels from their publicly available object detection implementations [140].

The Jetson Xavier AGX is equipped with a CPU, a GPU, and two NPUs (known as NVDLAs). In these experiments, we quantized both the MMDAPN models and YOLOv4 models to FP16 precision, as it offers improved latency without significant accuracy loss. While INT8 quantization is also possible through a calibration process, it can result in a notable decrease in accuracy due to the reduced numeric range compared to FP16 or FP32.

Figure 5.12 depicts the optimal pipeline configurations for our approach and JEDI. We only considered candidates that fell within the limits of Δ_{thr} and P_{thr}. Each candidate's hardware deployment configuration is represented by different shapes. For our approach, rectangles represent that all DNNs run on the GPU, triangles represent 4

DNNs on the GPU and 2 on the 2 NPUs (one on each), and circles indicate 5 DNNs on the GPU and 1 on a single NPU. Stars symbolize JEDI's optimized YOLOv4 for GPU configurations $k = 1,3,5,15$ and $x = 1,3,5,15$ that fall within the Δ_{thr} and P_{thr} limits. Since JEDI does not support multi-DNN execution, we considered $y = 1$ for DNN threading. The hexagon denotes JEDI's best YOLOv4 configuration, utilizing 2 NVDLAs and GPU hardware with the PND-A technique and a (82) cutpoint, employing a model configuration of $k = 1$, $x = 1$, and $y = 1$. The graph presents the latency in milliseconds and power consumption in Watts.

The experiment demonstrates that our method's optimal configuration (GPU2NPU _k15x1y6) outperforms JEDI's (YOLOv4GPU_k5x3y1): ~16ms/~15W compared to JEDI's ~20ms/~22W. Our method's optimal configuration uses the GPU and 2 NPUs for multi-DNN inference with $k = 15$, $x = 1$, and $y = 6$. This suggests that loading 15 frames to preprocess all crops in the same thread ($k = 15$, $x = 1$), but executing DNN inference in a separate thread, is more efficient ($y = 6$). Notably, 80% of the top 10 candidates employ a GPU-NPU hardware combination (8 out of 10). The experiment also reveals that using two threads for DNN ($y = 2$) yields the fastest latency results with GPU hardware configuration (GPU_k15x1y2). However, using $y = 3$ results in the worst performance across all GPU configurations, with higher latencies and power consumption. JEDI's YOLOv4 candidates for GPU configurations indicate that employing multiple preprocessing threads leads to faster inference results for $x = 5$ and $x = 15$, surpassing the MMDAPN approach's performance with a single thread for DNN ($y = 1$).

5.4 On-site readjustment of the system in an aircraft cabin

One of the main challenges in using DNNs for practical applications is their need for significant computational resources for fine-tuning when false positives (FPs) and negatives (FNs) are detected. This can be a problem when these resources are not available onboard, such as in aircrafts where internet connectivity for cloud computing is not always possible. As a result, alternative approaches must be considered for model readjustments.

Figure 5.13 shows our proposed workflow for this goal, where we consider the following components:

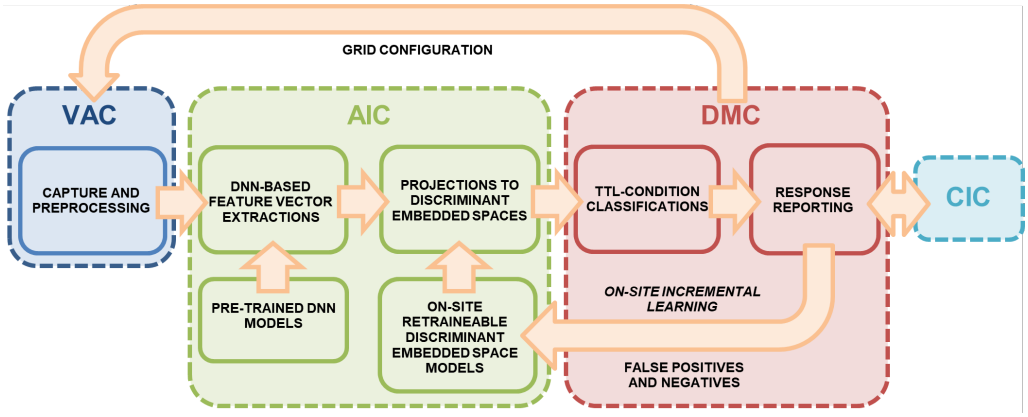


Figure 5.13: The workflow for multi-DNN-based onboard video analytics with on-site model readjustment.

1. **The video acquisition component (VAC)**: It is in charge of managing all aspects of video acquisition. This includes optimal and multithreaded camera capture, as well as image encoding and decoding functionalities for the source image. Additionally, VAC contains various image processing algorithms to prepare data for multi-DNN model inference.

2. **The AI component (AIC)**: It contains the algorithm to process the captured images for the TTL-condition classification. This operation is done in two stages: first, feature vectors are extracted with the MMDAPNs explained above, and then, these vectors are projected to discriminant embedded spaces (DES) modeled with manifold-learning-based approaches such as Linear Discriminant Analysis (LDA) [141]. The DES produces the effect of increasing inter-class and reducing intra-class feature distances.

3. **The data management component (DMC)**: It is the core component of the application. It processes the responses from the AIC to obtain the TTL-condition classification result, given a user-defined sensitivity threshold. Meanwhile, the user interface interacts with the DMC's features to manage the application's logic.

4. **The communication interface component (CIC)**: It manages the bi-directional communication between the AI-processor result information and the cabin video monitoring system (CVMS) of the aircraft.

The AIC's DES models are trained using representative feature vectors for each class. Training DES models such as LDA is faster and can be done on an embedded AI-processor such as the Jetson Xavier AGX. This allows for on-site readjustment with new samples, a process that we call on-site incremental learning (OSIL). To prevent model degradation during OSIL, quality checks are performed using a testing dataset. If a model fails to meet the required quality, it is not updated on-site. In that case the MMDAPN should be retrained off-site.

The CVMS interface, shown in Figure 5.14, allows for system readjustment using the OSIL approach. Signals from the Jetson board are received via Ethernet. The interface displays minimalistic graphics of the system's response for analyzed seats and two corridor sections on the left: green indicates a correct TTL situation, red indicates incorrect, and yellow indicates uncertain. If a failure is observed (Figure 5.14), the user can correct it by stopping the system using the central blue buttons, selecting the location of the failure and the correct class using the central green and red buttons, and then retraining and restarting the system using the central blue buttons.

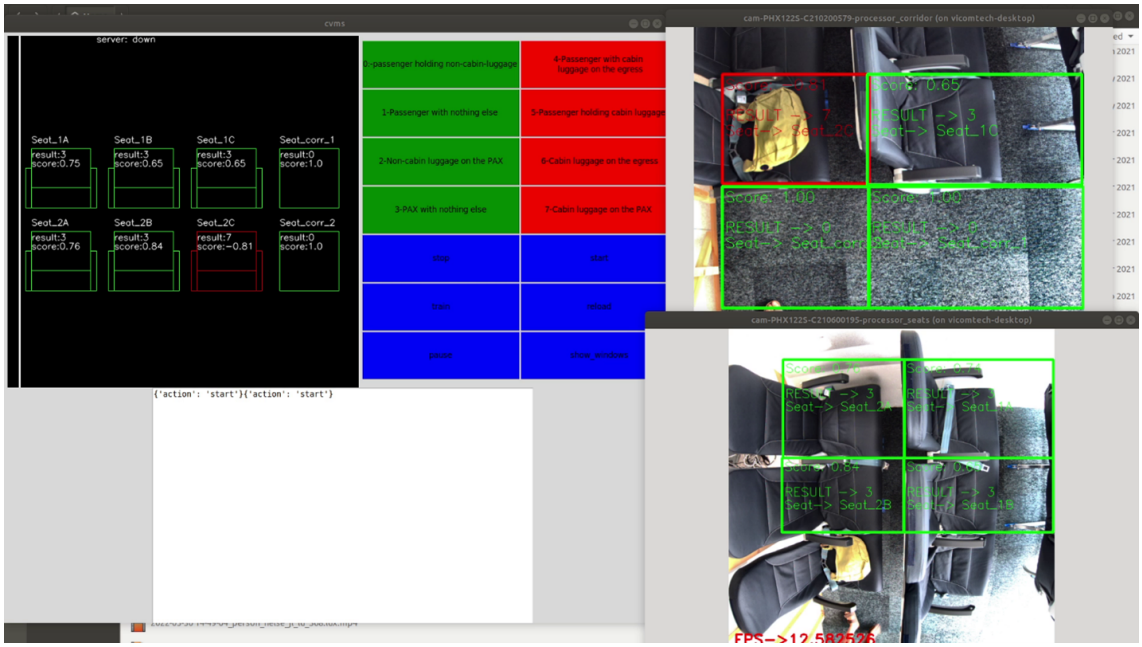


Figure 5.14: An example of the CVMS: On the left, the control panel and seat monitoring. On the right, the post-processing results.

5.4.1 Experimental results

To evaluate the OSIL procedure, we conducted additional experiments in a cabin mockup. Over one month, we recorded several trials with different passengers performing various use cases in each cabin seat. They wore different clothes and varied their body and object poses while being recorded. Throughout these recordings, the system was operational at all times. Then, an evaluator segmented the videos to count distinguishable trials and check the system's response. Whenever a FP or FN was detected, it was counted towards the accuracy, FP, and FN calculations.

These trials were conducted under three different illumination conditions: high (500-1500 lux), normal (250-450 lux), and low (10-100 lux). The cameras were set to automatically adjust the signal's gain to accommodate these varying illumination levels. As a result, the visual appearance of the captured images remained relatively consistent across all three conditions. The recordings were split into two testing sessions, each lasting two weeks.

During the first session, the system was tested without using OSIL. In the second session, the OSIL approach was applied to adjust the system whenever a FP or FN was detected. To evaluate the potential of OSIL, we then compared the differences observed between the two sessions.

The number of trials counted during the first session is summarized in Table 5.4. Here PAX refers to a passenger seat. Figure 5.5 shows the obtained average accuracy, FP, and FN results. We confirmed that illumination differences did not significantly impact the system's functioning. Only two cases had results below 90%: non-cabin luggage on the PAX (84.82%) and passenger with cabin luggage in the egress (87.04%). The average accuracy was 94.61%.

Table 5.4: Overview of the trials conducted in testing session 1.

Placement type	Class N	TTL-cond.	Passenger	Use-case description	N trials			Total
					High illum.	Normal illum.	Low illum.	
Normal seat	0	Correct	Yes	Passenger holding non-cabin luggage	39	39	39	117
	1			Passenger with nothing else	54	53	55	162
	2		No	Non-cabin luggage on the PAX	37	37	39	113
	3			PAX with nothing else	62	60	60	182
	4	Incorrect	Yes	Passenger with cabin luggage on the egress	36	36	36	108
	5			Passenger holding cabin luggage	33	37	39	109
	6		No	Cabin luggage on the egress	37	37	36	110
	7			Cabin luggage on the PAX	40	41	40	121
Corridor	8	Correct	No	Corridor section without luggage	20	20	20	60
	9	Incorrect		Corridor section with luggage	12	12	10	34
Exit row seat	10	Incorrect	Yes	Passenger with luggage under the front seat in the exit row	17	18	18	53
	11	Incorrect	No	Luggage under the front seat in the exit row	18	18	18	54
				TOTAL	405	408	410	1,223

Table 5.5: Summary of the results obtained in testing session 1.

Placement type	Class N	TTL-cond.	Passenger	Use-case description	TTL-cond. Acc. (%) w/o uncertain and difficult cases
Normal seat	0	Correct	Yes	Passenger holding non-cabin luggage	97.44
	1			Passenger with nothing else	91.96
	2		No	Non-cabin luggage on the PAX	84.82
	3			PAX with nothing else	98.92
	4	Incorrect	Yes	Passenger with cabin luggage on the egress	87.04
	5			Passenger holding cabin luggage	93.57
	6		No	Cabin luggage on the egress	99.07
	7			Cabin luggage on the PAX	97.52
Corridor	8	Correct	No	Corridor section without luggage	100.00
	9	Incorrect		Corridor section with luggage	100.00
Exit row seat	10	Incorrect	Yes	Passenger with luggage under the front seat in the exit row	90.52
	11	Incorrect	No	Luggage under the front seat in the exit row	94.44
				AVERAGE	94.61

The observed main reasons for failure in this session were the following:

- non-cabin cabin luggage on the PAX vs Passenger holding cabin luggage: Jackets are confused with back bags and vice versa.
- Luggage under the front seat in the exit row: A small portion of the object is visible in some cases.
- Passenger with cabin luggage in the egress: A small portion of the object is visible in some cases.

In contrast, Table 5.6 summarizes the number of trials counted during the second session. Table 5.7 shows the average accuracy, FP, and FN results. In this case, all cases are above 90%, and the average accuracy is 95.74%. Therefore, this demonstrates that OSIL has improved the system's functioning.

Table 5.6: Overview of the trials conducted in testing session 2.

Placement type	Class N	TTL-cond.	Passenger	Use-case description	N trials			Total
					High illum.	Normal illum.	Low illum.	
Normal seat	0	Correct	Yes	Passenger holding non-cabin luggage	108	48	64	220
	1			Passenger with nothing else	36	54	13	103
	2		No	Non-cabin luggage on the PAX	139	104	83	326
	3			PAX with nothing else	98	71	43	212
	4	Incorrect	Yes	Passenger with cabin luggage on the egress	18	48	35	101
	5			Passenger holding cabin luggage	38	38	98	174
	6		No	Cabin luggage on the egress	54	45	30	129
	7			Cabin luggage on the PAX	83	106	51	240
Corridor	8	Correct	No	Corridor section without luggage	37	26	22	85
	9	Incorrect		Corridor section with luggage	26	16	20	62
Exit row seat	10	Incorrect	Yes	Passenger with luggage under the front seat in the exit row	15	34	42	91
	11	Incorrect	No	Luggage under the front seat in the exit row	37	24	42	103
				TOTAL	689	614	543	1,846

Table 5.7: Summary of the results obtained in testing session 2.

Placement type	Class N	TTL-cond.	Passenger	Use-case description	TTL-cond. Acc. (%) w/o uncertain and difficult cases
Normal seat	0	Correct	Yes	Passenger holding non-cabin luggage	92.15
	1			Passenger with nothing else	92.19
	2		No	Non-cabin luggage on the PAX	96.07
	3			PAX with nothing else	98.29
	4	Incorrect	Yes	Passenger with cabin luggage on the egress	97.22
	5			Passenger holding cabin luggage	93.23
	6		No	Cabin luggage on the egress	96.91
	7			Cabin luggage on the PAX	93.34
Corridor	8	Correct	No	Corridor section without luggage	100.00
	9	Incorrect		Corridor section with luggage	100.00
Exit row seat	10	Incorrect	Yes	Passenger with luggage under the front seat in the exit row	96.80
	11	Incorrect	No	Luggage under the front seat in the exit row	92.64
				AVERAGE	95.74

5.5 Conclusions

In this work, we presented an approach for verifying the proper placement of luggage in aircraft cabins by deploying on-board video analytics with an optimal balance between response delay and power consumption. Our approach combines metric-guided multi-task domain adversarial prototypical Networks (MMDAPNs) with an optimized processing pipeline that takes advantage of the heterogeneous computing capabilities of embedded processors. To assess the effectiveness of our approach, we conducted experiments using the SmaCS dataset [134], which was specifically created for research in this domain.

Experimental results demonstrate that the MMDAPN architecture significantly enhances the classification accuracy of overall luggage positioning correctness when compared to existing state-of-the-art alternatives. This notable improvement can be attributed to the incorporation of a subclasses hierarchy, along with the matrix-guided prototypical component, which effectively influences the arrangement of subclasses within the feature space.

In contrast to alternative state-of-the-art multi-DNN processing pipelines, our approach harnesses the heterogeneous computing capabilities available and effectively determines the optimal configuration that strikes a balance between processing latency and power consumption. To evaluate these capabilities, we conducted an end-to-end deployment of the multi-MMDAPN system and compared it to an equivalent state-of-the-art approach for object detection [140] on the Jetson AGX platform. Our findings indicate that employing higher batch sizes and increasing the number of worker DNN

threads contribute to more efficient pipeline processing in both scenarios. Furthermore, we observed that the simultaneous multi-DNN classification approach outperforms object detection in terms of optimization and efficiency.

In future work, we plan to investigate incremental learning techniques that enable the system to adapt and readjust on-site using the embedded processor in response to false positives and negatives. Additionally, we will explore methods to incorporate spatio-temporal features into the system without compromising its performance, aiming to enhance its accuracy and robustness further.

Part IV

Conclusions

CHAPTER

6

Conclusions and future work

6.1 Conclusions

This thesis studied the key factors for the optimal deployment of DNN-based vision applications in diverse and heterogeneous intelligent system infrastructures. The main objective was to generate and optimally deploy a modular processing pipeline to improve performance indices. More specifically, this work proposes different strategies to effectively address optimal deployment issues considering the needs of intelligent system infrastructures, tailored to the hardware/software specifications and adjusted to the current DNN inference engines and optimizations. Our concluding observations and the corresponding strategies proposed in this research work are the following:

Observation 1: A serverless cloud computing model is a suitable alternative for deploying end-to-end DNN-based vision applications at high scale.

Strategy 1: The optimal management of the allocated memory of serverless function instances is a crucial factor to maximize the DNN processing performance at minimum cost. Furthermore, the bottleneck resides in the DNN processing task of each serverless function instance. While lightweight DNN models can be deployed effectively in serverless environments, deploying large DNN models is not currently feasible due to serverless limitations.

In Chapter 3, we proposed two contributions to define these strategies. The first approach assesses the DNN performance in serverless environments. Based on the MLPerf standard guidelines, this contribution presents a novel decomposition methodology for benchmarking DNN inference in a serverless function execution model. Chapter 3 also proposed an approach to optimally deploy several DNN models to FaaS platforms supported by the latest computer vision techniques. Through different experimental studies, we conclude that defining optimal memory allocation for each function influences the inference latency and economic cost. In particular, the allocated memory amount performed best between 768MB and 1536MB in our set of experiments. Also, the OpenVINO IE DNN inference engine and OpenVINO IR model optimizations contribute to reduce the inference latency in Amazon Lambda hardware. Finally, the research results indicate that DNN processing latency has negligible influence on throughput results providing 51-83 queries per second in Amazon Lambda.

Observation 2: Deploying DNN-based vision applications on IoT platforms requires dealing with device hardware heterogeneity and DNN inference optimization. But it also requires focusing on the optimal processing of other components such as pre-processing and user interaction while preserving privacy and enhancing security.

Strategy 2: Create deployment strategies tailored to the target hardware and software specificities (operating system (OS), RAM, AI accelerators), focused on the DNN environment (number of DNN models, DNN model configurations, task delegation strategies), and, designed to end-to-end ISVA application requirements (encryption algorithm, network balance, and user interaction workflow).

In chapter 4 we described a knowledge-driven approach for the optimal deployment of DNN-based face recognition solutions in a heterogeneous IoT platform for older adult care and industrial applications. This methodology tackles its specific challenges in terms of ease of use, hardware heterogeneity, and security. Ease of use is covered by a customized workflow that offers an intuitive interface and considers the shared use between the user and companions, by means of automated feedback to automatically take the “selfie” with the best possible quality. Device heterogeneity is addressed by a smart deployment optimizer, capable of selecting the appropriate DNN inference approach for the targeted device’s hardware specifications. Data security is enforced by a scheme that avoids the transmission and storage of biometric data in the cloud and employs fully homomorphic encryption to perform all face matching operations directly on the

encrypted domain. A qualitative comparison and preliminary experiments to select the optimal inference approach for different computing situations (different batch sizes and computing noise) reveal its potential for the desired goal.

Observation 3: Optimally deploying DNN-based vision applications on edge video analytic pipelines to maximize the hardware usage while minimizing power consumption demands the optimization of the whole processing pipeline. This requires a deep analysis of DNN model optimizations, pre- and post- image processing algorithms, resource scheduling, and parallelization strategies.

Strategy 3: To effectively address this observation, the following key factors should be taken into consideration: (1) Identify the ISVA end-to-end vision application requirements and performance metrics to design the processing pipeline, (2) Create an optimal scheduling method for the video analytics pipeline that maximizes the heterogeneous hardware capabilities for DNN inference, reduces the complexity of DNN models, minimizes processing latency, and maintains prediction accuracy.

In Chapter 5 we present an end-to-end approach for deploying a multi-DNN-based on-board video analytics system for edge computing environments. This approach takes the verification of the correct positioning of luggage in aircraft as a use case scenario. More specifically, this approach is done by deploying onboard video analytics that balances response time and power consumption. This methodology uses metric-guided multi-task domain adversarial prototypical Networks (MMDAPNs) and an optimal processing pipeline. This leverages the heterogeneous computing capabilities of the embedded processor. We evaluated our approach using the SmaCS dataset [134]. The pipeline processing algorithm automatically finds the best balance between latency and power consumption. To enable onboard system adjustments, MMADPNs can serve as feature extractors that are projected onto retrainable discriminant embedded spaces modeled using manifold-learning approaches. Based on the experiments we conclude that larger batch sizes and more worker DNN threads are more efficient for optimal pipeline processing.

6.2 Future work

Serverless computing's high scaling capabilities offer a bright future for ISVA vision applications and services. However, to overcome serverless computing limitations, further

research is needed to reduce DNN model and computation complexity. One potential solution is to distribute DNN processing into multi-tenant systems. The design space is vast, with different serverless environments, benchmarks, and hardware targets requiring further investigation. We plan to expand our benchmarking evaluations to popular serverless function platforms and explore benchmarking more complex ML systems that consider a computing continuum formed by mobile, edge, and cloud resources [106], using standards such as MLPerf.

As advancements in DNN architectures and optimizations continue, the diverse nature of intelligent system infrastructure is likely to present more challenges and issues. For example, there is a growing interest in research on explainable AI systems, which requires the integration of additional processes to interpret the results of DNN black-boxes. This can pose a challenge for real-time systems. Therefore, it is essential to conduct further experiments with a broader range of users and equipment and to study the interaction issues with these techniques in greater depth. Knowledge-driven approaches may be helpful in addressing these challenges.

The research and developer community is suggesting guidelines for the deployment life-cycle of ML, from training to inference processing in the context of MLOps. These guidelines focus on deploying conventional object detection and classification results for computer vision problems in ISVA systems. As AI hardware, DNN deployment tools, and architectures continue to develop, new approaches for standardizing deployment strategies will be proposed by the research community for the MLOps community.

Part V

Appendix

APPENDIX

A

Publications related to the research done for this thesis

A.1 Benchmarking deep neural network inference performance on serverless environments with MLPerf

Title: Benchmarking deep neural network inference performance on serverless environments with mlperf

Authors: Unai Elordi, Luis Unzueta, Jon Goenetxea, Sergio Sanchez-Carballido, Ignacio Arganda-Carreras, Oihana Otaegui

Journal: IEEE SOFTWARE

Year: 2021

DOI: `https://.doi.org/10.1109/MS.2020.3030199`

Abstract: *We provide a novel decomposition methodology from the current MLPerf benchmark to the serverless function execution model. We have tested our approach in Amazon Lambda to benchmark the processing capabilities of OpenCV and OpenVINO inference engines.*

A.2 Designing automated deployment strategies of face recognition solutions in heterogeneous iot platforms.

Title: Designing automated deployment strategies of face recognition solutions in heterogeneous iot platforms.

Authors: Unai Elordi, Chiara Lunerti, Luis Unzueta, Jon Goenetxea, Nerea Aranjuelo, Alvaro Bertelsen, Ignacio Arganda-Carreras

Journal: Information

Year: 2021

DOI: `https://doi.org/10.3390/info12120532`

Abstract: *In this paper, we tackle the problem of deploying face recognition (FR) solutions in heterogeneous Internet of Things (IoT) platforms. The main challenges are the optimal deployment of deep neural networks (DNNs) in the high variety of IoT devices (e.g., robots, tablets, smartphones, etc.), the secure management of biometric data while respecting the user's; privacy, and the design of appropriate user interaction with facial verification mechanisms for all kinds of users. We analyze different approaches to solving all these challenges and propose a knowledge-driven methodology for the automated deployment of DNN-based FR solutions in IoT devices, with the secure management of biometric data, and real-time feedback for improved interaction. We provide some practical examples and experimental results with state-of-the-art DNNs for FR in Intel's and NVIDIA's hardware platforms as IoT devices.*

A.3 Optimal deployment of face recognition solutions in a heterogeneous iot platform for secure elderly care applications.

Title:Optimal deployment of face recognition solutions in a heterogeneous iot platform for secure elderly care applications.

Authors: Unai Elordi, Alvaro Bertelsen, Luis Unzueta, Nerea Aranjuelo, Jon Goenetxea, Ignacio Arganda-Carreras

Proceedings: Procedia Computer Science

Year: 2021
DOI: https://doi.org/10.1016/j.procs.2021.09.093
Abstract: *Face recognition provides a desirable solution for authentication and surveillance in Internet of Things platforms for elderly care. However, its inclusion is challenging because of the possibly reduced interaction capabilities of users, the high variety of interaction devices, and the need of managing biometric data securely. Our approach relies on lightweight deep neural networks for secure recognition and to guide users during interaction. An automated procedure selects the appropriate inference engine, model configurations, and batch size, based on edge device characteristics. Biometric data is homomorphically encrypted to preserve privacy. An evaluation with respect to state-of-the-art alternatives shows its potential.*

A.4 On-demand Serverless Video Surveillance with Optimal Deployment of Deep Neural Networks.

On-demand Serverless Video Surveillance with Optimal Deployment of Deep Neural Networks.
Authors: Unai Elordi, Luis Unzueta, Jon Goenetxea, Estíbaliz Loyo, Ignacio Arganda-Carreras, Oihana Otaegui
Proceedings: VISIGRAPP (4: VISAPP)
Pages: 717-723
publisher: SciTePress
organization: INSTICC
isbn: 978-989-758-488-6
issn: 2184-4321
Year: 2021
DOI: https://doi.org/10.5220/0010344807170723
Abstract: *We present an approach to optimally deploy Deep Neural Networks (DNNs) in serverless cloud architectures. A serverless architecture allows running code in response to events, automatically managing the required computing resources. However, these resources have limitations in terms of execution environment (CPU only), cold starts,*

space, scalability, etc. These limitations hinder the deployment of DNNs, especially considering that fees are charged according to the employed resources and the computation time. Our deployment approach is comprised of multiple decoupled software layers that allow effectively managing multiple processes, such as business logic, data access, and computer vision algorithms that leverage DNN optimization techniques. Experimental results in AWS Lambda reveal its potential to build cost-effective on-demand serverless video surveillance systems.

A.5 Leveraging Synthetic Data for DNN-Based Visual Analysis of Passenger Seats.

Leveraging Synthetic Data for DNN-Based Visual Analysis of Passenger Seats

Authors: Nerea Aranjuelo, Jose Luis Apellaniz, Luis Unzueta, Jorge García, Sara García, Unai Elordi and Oihana Otaegui

Journal: SN Computer Science 4

issn: 2661-8907

Year: 2023

DOI: `https://doi.org/10.1007/s42979-022-01453-x`

Abstract: *Deep neural network (DNN)-based vision systems could improve passenger transportation safety by automating processes such as verifying the correct positioning of luggage, seat occupancy, etc. Abundant and well-distributed data are essential to make DNNs learn appropriate pattern recognition features and have enough generalization ability. The use of synthetic data can reduce the effort of generating varied and annotated data. However, synthetic data usually present a domain gap with real-world samples, that can be reduced with domain adaptation techniques. This paper proposes a methodology to build simulated environments to generate balanced and varied synthetic data and avoid including redundant samples to train classification DNNs for passenger seat analysis. We show a practical implementation for detecting whether luggage is correctly placed or not in an aircraft cabin. Experimental results show the contribution of the synthetic samples and the importance of correctly discarding redundant data.*

A.6 Building Synthetic Simulated Environments for Configuring and Training Multi-camera Systems for Surveillance Applications

uilding Synthetic Simulated Environments for Configuring and Training Multi-camera Systems for Surveillance Applications

Authors: Unai Elordi, Luis Unzueta, Jon Goenetxea, Estíbaliz Loyo, Ignacio Arganda-Carreras, Oihana Otaegui

Proceedings: VISIGRAPP (5: VISAPP)

Pages: 80-91

publisher: SciTePress

organization: INSTICC

isbn: 978-989-758-488-6

issn: 2184-4321

Year: 2021

DOI: https://doi.org/10.5220/0010344807170723

Abstract: *Synthetic simulated environments are gaining popularity in the Deep Learning Era, as they can alleviate the effort and cost of two critical tasks to build multi-camera systems for surveillance applications: setting up the camera system to cover the use cases and generating the labeled dataset to train the required Deep Neural Networks (DNNs). However, there are no simulated environments ready to solve them for all kind of scenarios and use cases. Typically, 'ad hoc' environments are built, which cannot be easily applied to other contexts. In this work we present a methodology to build synthetic simulated environments with sufficient generality to be usable in different contexts, with little effort. Our methodology tackles the challenges of the appropriate parameterization of scene configurations, the strategies to generate randomly a wide and balanced range of situations of interest for training DNNs with synthetic data, and the quick image capturing from virtual cameras considering the rendering bottlenecks. We show a practical implementation example for the detection of incorrectly placed luggage in aircraft cabins, including the qualitative and quantitative analysis of the data generation process and its influence in a DNN training, and the required modifications to adapt it to other surveillance contexts.*

A.7 Building a Camera-based Smart Sensing System for Digitalized On-demand Aircraft Cabin Readiness Verification

Building a Camera-based Smart Sensing System for Digitalized On-demand Aircraft Cabin Readiness Verification

Authors: Luis Unzueta, Sandra Garcia, Jorge Garcia, Valentin Corbin, Nerea Aranjuelo, Unai Elordi, Oihana Otaegui, Maxime Danielli

Proceedings: Proceedings of the International Conference on Robotics, Computer Vision and Intelligent Systems - ROBOVIS

Pages: 98-105

publisher: SciTePress

organization: INSTICC

isbn: 978-989-758-479-4

Year: 2021

DOI: https://doi.org/10.5220/0010128500980105

Abstract: *Synthetic simulated environments are gaining popularity in the Deep Learning Era, as they can alleviate the effort and cost of two critical tasks to build multi-camera systems for surveillance applications: setting up the camera system to cover the use cases and generating the labeled dataset to train the required Deep Neural Networks (DNNs). However, there are no simulated environments ready to solve them for all kind of scenarios and use cases. Typically, 'ad hoc' environments are built, which cannot be easily applied to other contexts. In this work we present a methodology to build synthetic simulated environments with sufficient generality to be usable in different contexts, with little effort. Our methodology tackles the challenges of the appropriate parameterization of scene configurations, the strategies to generate randomly a wide and balanced range of situations of interest for training DNNs with synthetic data, and the quick image capturing from virtual cameras considering the rendering bottlenecks. We show a practical implementation example for the detection of incorrectly placed luggage in aircraft cabins, including the qualitative and quantitative analysis of the data generation process and its*

influence in a DNN training, and the required modifications to adapt it to other surveillance contexts.

A.8 How can deep neural networks be generated efficiently for devices with limited resources?

How can deep neural networks be generated efficiently for devices with limited resources?

Authors: Unai Elordi, Luis Unzueta, Ignacio Arganda-Carreras, Oihana Otaegui

Proceedings: Articulated Motion and Deformable Objects - 10th International Conference

Pages: 4-33

publisher: Springer International Publishing

isbn: 978-3-319-94544-6

Year: 2018

DOI: `https://doi.org/10.5220/0010128500980105`

Abstract: *Despite the increasing hardware capabilities of embedded devices, running a Deep Neural Network (DNN) in such systems remains a challenge. As the trend in DNNs is to design more complex architectures, the computation time in low-resource devices increases dramatically due to their low memory capabilities. Moreover, the physical memory used to store the network parameters augments with its complexity, hindering a feasible model to be deployed in the target hardware. Although a compressed model helps reducing RAM consumption, a large amount of consecutive deep layers increases the computation time. Despite the wide literature about DNN optimization, there is a lack of documentation for practical and efficient deployment of these networks. In this paper, we propose an efficient model generation by analyzing the parameters and their impact and address the design of a simple and comprehensive pipeline for optimal model deployment.*

A.9 Optimizing Video Analytics Deployment for In-Flight Cabin Readiness Verification

Optimizing Video Analytics Deployment for In-Flight Cabin Readiness Verification

Authors: Unai Elordi, Nerea Aranjuelo, Luis Unzueta, Jose Luis Apellaniz, Ignacio Arganda-Carreras

Journal: IEEE access

DOI: `https://doi.org/10.5220/0010128500980105`

Abstract: *In this paper, we propose an approach to optimize the deployment of on-board video analytics for checking the correct positioning of luggage in aircraft cabins. The system consists of embedded cameras installed on top of the cabin and a heterogeneous embedded processor. Each camera covers multiple regions of interest (i.e., multiple seats or aisle sections) to minimize the number of cameras required. Each image region is processed by a separate image classification algorithm trained with the expected kind of visual appearance considering the effect of perspective and lens distortion. They classify these regions as correct or incorrect for cabin readiness by exploiting the hierarchical structure of classes composed of different configurations of passengers' and objects' presence or absence and the objects' location. Our approach leverages semantic distances between classes to guide prototypical neural networks for multi-tasking between the main classification (i.e., correct or incorrect status) and auxiliary attributes (i.e., scene configurations), learning robust features from different data domains (i.e., various cabins, real or synthetic). The processing pipeline optimizes response delay and power consumption by leveraging embedded processors' computing capabilities. We carried out experiments in a cabin mockup with a Jetson AGX Xavier, efficiently obtaining better-quality descriptive information from the scene to improve the system's accuracy compared to alternative state-of-the-art methods.*

A.10 Multi-Task Explainable Quality Networks for Large-Scale Forensic Facial Recognition

Multi-Task Explainable Quality Networks for Large-Scale Forensic Facial Recognition

Authors: Andrea Macarulla Rodriguez, Luis Unzueta and Zeno Geradts, Marcel Worring, Unai Elordi

Journal: IEEE Journal of Selected Topics in Signal Processing

Pages:1-12 **Year:** 2023

DOI: https://doi.org/10.1109/JSTSP.2023.3267263

Abstract: *Identifying suspects from surveillance footage is a crucial task in forensic investigations, but it is often hindered by the variable conditions of observation and the large amounts of data. Face image quality (FIQ) is a metric that measures the usefulness of a face sample for facial recognition. Existing methods for automated FIQ assessment only provide a scalar value for quality, and do not indicate which factors are causing low quality. Additionally, these methods are computationally expensive, which makes current FIQ assessment methods unsuitable for large numbers of images. To address these issues, we introduce multi-task explainable quality networks (XQNets). XQNets provide both the quality value and the associated facial and environmental attributes, and automatically learn how each attribute contributes to the quality value during the training process. We also propose a dataset-agnostic quality pairing protocol to ensure that sample pairs are balanced across datasets and evaluations are fair. Our experiments on the LFW and SC-face benchmarks show that our approach generalizes well across different datasets and outperforms state-of-the-art methods. Our method offers a fast, explainable approach to FIQ assessment, making it suitable for large-scale forensic applications.*

APPENDIX

B

Other publications related with the application of computer vision and Deep Neural Networks field

B.1 A temporally consistent grid-based visual odometry framework for multi-core architectures.

Title: A temporally consistent grid-based visual odometry framework for multi-core architectures.

Authors: Leonardo de Maeztu, Unai Elordi, Marcos Nieto, Javier Barandiarán, Oihana Otaegui

Journal: Journal of Real-Time Image Processing 10

Year: 2015

Pages: 59–769

DOI: https://doi.org/10.1007/s11554-014-0425-y

Abstract: *Most recent visual odometry algorithms based on sparse feature matching are*

computationally efficient methods that can be executed in real time on desktop computers. However, further efforts are required to reduce computational complexity in order to integrate these solutions in embedded platforms with low power consumption. This paper presents a spacetime framework that can be applied to most stereo visual odometry algorithms greatly reducing their computational complexity. Moreover, it enables exploiting multi-core architectures available in most modern computing platforms. According to the tests performed on publicly available datasets and an experimental driverless car, the proposed framework significantly reduces the computational complexity of a visual odometry algorithm while improving the accuracy of the results.

B.2 **Efficient Multi-task based Facial Landmark and Gesture Detection in Monocular Images.**

Efficient Multi-task based Facial Landmark and Gesture Detection in Monocular Images

Authors: Jon Goenetxea, Luis Unzueta, Unai Elordi, Oihana Otaegui, Fadi Dornaika
Proceedings: VISIGRAPP (5: VISAPP)
Pages: 680-687
publisher: SciTePress
organization: INSTICC
isbn: 978-989-758-488-6
issn: 2184-4321
Year: 2021
DOI: https://doi.org/110.5220/0010373006800687

Abstract: *The communication between persons includes several channels to exchange information between individuals. The non-verbal communication contains valuable information about the context of the conversation and it is a key element to understand the entire interaction. The facial expressions are a representative example of this kind of non-verbal communication and a valuable element to improve human-machine interaction interfaces. Using images captured by a monocular camera, automatic facial analysis systems can extract facial expressions to improve human-machine interactions. However,*

there are several technical factors to consider, including possible computational limitations (e.g. autonomous robots), or data throughput (e.g. centralized computation server). Considering the possible limitations, this work presents an efficient method to detect a set of 68 facial feature points and a set of key facial gestures at the same time. The output of this method includes valuable information to un derstand the context of communication and improve the response of automatic human-machine interaction systems.

B.3 **Virtual reality interfaces applied to web-based 3D E-commerce.**

EVirtual reality interfaces applied to web-based 3D E-commerce
Authors: Unai ELordi, Álvaro Segura, Jon Goenetxea,
Proceedings: VISIGRAPP (5: VISAPP)
Pages: 680-687
publisher: SciTePress
organization: INSTICC
isbn: 978-989-758-488-6
issn: 2184-4321
Year: 2021
DOI: `https://doi.org/110.5220/0010373006800687`
Abstract: *The communication between persons includes several channels to exchange information between individuals. The non-verbal communication contains valuable information about the context of the conversation and it is a key element to understand the entire interaction. The facial expressions are a representative example of this kind of non-verbal communication and a valuable element to improve human-machine interaction interfaces. Using images captured by a monocular camera, automatic facial analysis systems can extract facial expressions to improve human-machine interactions. However, there are several technical factors to consider, including possible computational limitations (e.g. autonomous robots), or data throughput (e.g. centralized computation server). Considering the possible limitations, this work presents an efficient method to detect a set of 68 facial feature points and a set of key facial gestures at the same time. The output of*

this method includes valuable information to un derstand the context of communication and improve the response of automatic human-machine interaction systems.

B.4 Efficient Multi-task based Facial Landmark and Gesture Detection in Monocular Images.

Efficient Multi-task based Facial Landmark and Gesture Detection in Monocular Images

Authors: Alejandro Clemotte , Harbil Arregui, Miguel A. Velasco, Luis Unzueta, Jon Goenetxea, Unai Elordi, Eduardo Rocón, Ramón Ceres, Javier Bengoechea, Iosu Arizkuren ,Eduardo Jauregui

Proceedings: Actas de las XXXVII Jornadas de Automática

Pages: 680-687

publisher: SciTePress

organization: INSTICC

isbn: 978-84-9749-808-1

Year: 150-155

DOI: `https://doi.org/110.5220/0010373006800687`

Abstract: *TThis paper presents a pilot study completed in the framework of the INTER-AAC project. The aim of the project is to develop a new human-computer interaction (HCI) solution based on eye-gaze estimation from webcam images for people with motor disorders such as cerebral palsy, neurodegenerative diseases, and spinal cord injury that are otherwise unable to use a keyboard or mouse. In this study, we analyzed cursor trajectories recorded during the experiment and validated that users with different diseases can be automatically classi ed in groups based on trajectory metrics. For the clustering, Ward's method was used. The metrics are based on speed and acceleration statistics from full fi ltered tracks. The results show that the participants can be grouped into two main clusters. The main contribution of this work is the evaluation of the clustering techniques applied to eye-gaze trajecto- ries for the automatic classi cation of users diseases based on a real experiment carried with the help of three clinical partners in Spain.*

APPENDIX

C

Patent applications

C.1 Method, System and Computer Program Product for Eye Gaze Direction Estimation

Abstract: *A computer-implemented method for estimating eye gaze direction, comprising: fitting (11) a 3D face model (111) to a monocular image (101) obtained from an imaging device, thus obtaining values of a set of face model parameters representing at least one model position parameter (t), at least one orientation parameter (r), at least one shape parameter (s) and at least one action parameter (a), obtaining (12) normalized 3D gaze estimation vectors for the right and left eyes (121) with respect to the imaging device viewpoint; estimating (13) the eye gaze direction with respect to at least one target in the scene. A system comprising at least one processor configured to perform the steps of the method. A computer program product comprising computer program instructions/code for performing the method. A computer-readable memory/medium that stores program instructions/code for performing the method.*

Applicants: Fundacion Centro De Tecnologias De Interaccion Visual Y Comunicaciones Vicomtech

Inventors: Unzueta Irurtia Luis [es]; Goenetxea Imaz Jon [es]; Elordi Hidalgo Unai [es]; Otaegui Madurga Oihana [es]

Application number: EP3506149A1
Application data: 2017-12-27
Publication data: 2019-07-03

C.2 Method and System for Detecting Presence of Objects in Passenger Compartments of Transport Means

Abstract: computer-implemented method for detecting presence of objects in passenger compartments of transport means, comprising: receiving, at a computing system, at least one image of the inside of a passenger compartment of a transport means; extracting, by a DNN, a descriptive feature vector associated to the received image, the descriptive feature vector containing information of the image indicative of the presence of objects within the passenger compartment, the objects being unallowed objects or objects located in unallowed locations within the passenger compartment; classifying, by a discriminant embedding space-based classifier, the image in a particular class of a set of predefined classes based on the associated descriptive feature vector of the image; and determining, by the computing system, the presence of unallowed objects or objects located in unallowed locations within the passenger compartment according to the corresponding class associated to the image.
Applicants: Fundacion Centro De Tecnologias De Interaccion Visual Y Comunicaciones Vicomtech
Inventors: Nerea Aranjuelo [es], Luis Unzueta [es], Unai Elordi [es], Jose Luis apellaniz [es], Sara García [es], Jorge García [es], Oihana Otaegui [es], Maxime Danielli [fr].
Application number: EP22382620
Application data: not available publicly
Publication data: review pending

APPENDIX

D

Other published resources

D.1 SmaCS dataset

Abstract: *The SmaCS dataset is created to support research on object detection and scene understanding, specifically related to identifying the proper positioning of cabin luggage during taxi, take-off, and landing (TTL) operations. Examples of cabin luggage include backpacks, bottles, briefcases, camcorders, hats, laptops, shopping bags, suitcases, tote bags, etc. During TTL, these items should not be placed in areas that could compromise the safety, such as on seats, egresses, held by passengers, or in the aisle. On the other hand, other types of objects, such as magazines, books, food, smartphones, tablets, jackets, and wallets, do not have these restrictions and should be ignored. To collect the necessary data, a cabin mockup was built, and synthetic data was generated using 3D graphics. The mockup features one side of three cabin seat rows, an aisle, and an exterior made of polystyrene. The mockup was illuminated from three possible light sources: natural light from the room's windows, artificial light on the ceiling, and a spotlight beside the cabin window to mimic directional sunlight. The recordings were done on different days under varying lighting conditions. We utilized two cameras to observe two seat rows, one placed above the four seats closest to the window, and another above the aisle, to monitor the aisle and the two seats beside it. We established a recording protocol that 18 participants followed to simulate a variety of situations with cabin and non-cabin luggage objects.*

OPTIMIZING DEEP NEURAL NETWORK DEPLOYMENT FOR INTELLIGENT SECURITY VIDEO ANALYTICS

The protocol instructed passengers on actions such as how long to sit in a specific seat and where to place carried items. The goal of the protocol was to create a balanced number of scenarios, such as correctly and incorrectly placed luggage. Here, we make public the data of one of the seats (closest to the cabin window of the back row). For this seat, we obtained around 34K real and 7K synthetic samples.

Applicants: FUNDACION CENTRO DE TECNOLOGIAS DE INTERACCION VISUAL Y COMUNICACIONES VICOMTECH

Inventors: Unai ELordi, Nerea Aranjuelo, Luis Unzueta, Jose Luis Apellaniz, Jorge Garcia, Oihana Otaegui.

DOI ttps://doi.org/10.5281/zenodo.7524808

Publication data: 2023-01-11

APPENDIX

E

Glossary

Acronyms

AI Artificial Intelligence

AIoT Artificial Intelligence Internet of Things

API Application Programming Interface

ASIC Application-specific Integrated Circuits

AU Atomic Unit

AWS Amazon Web Services

BNN Binary Neural Network

CBR Case Based Reasoning

CNN Convolutional Neural Network

CPU Central Processing Unit

CVMS Cabin Video Monitoring System

DES Discriminant Embedding Space

DL Deep Learning

DNN Deep Neural Networks

EBP Expectation Back-Propagation

FaaS Function as a Service

FD Face Detection

FIQ Face Image Quality

FIR Facial Identity Recognition

FLD Face Landmark Detection

FLOP Floating Point Operation

FN False Negative

FP False Positive

FPGA Field Programmable Gate Arrays

GDPR General Data Protection Regulation

GPU Graphic Processing Unit

HDDL High Density Deep Learning

IE Inference Engine

ILSVRC ImageNet Large Scale Visual Recognition Challenge

IoT Internet of Things

IQA Image Quality Assesment

IR Intermediate Representation

ISVA Intelligent Security Video Analyitcs

LBP Local Binary Patterns

MEC Multi-access Edge Computing

ML Machine Learning

MLOps Machine Learning Operations

MLTRL Machine Learning Technology Readinness level

MMDAPN Multitask Metric-guided Domain Adversarial Prototype Network

NS Notification Service

NVDLA NVidia Deep Learning Accelerator

ONNX Open Neural Network eXchange

OSIL On-Site Incremental Learning

OSS Online Storage System

PCA Principal Component Analysis

PGR Pose and Gesture Recognition

REST Representational State Transfer

RGBD Red Green Blue Depth

ROI Region Of Interest

ROS Robot Operating System

SAD Spoofing Attack Detection

SIMD Single Input Multiple Output

SLA Service Level Agreement

SLO Nueral Processing Unit

SLO Service Level Objectives

SM Streaming Multiprocessors

SNS Service Notification Service

SSD Single Shot Detection

SVD Singular Value Decomposition

TEE Trusted Execution Environment

TFLOP Tera Floating Point Operation

TPM Trust Platform Module

TPU Tensor Processing Unit

TRL Technology Readiness Level

TTL Taxi Take-off Landing

VGG Visual Geometry Group

ViT Vision Transformers

VPC Virtual Private Cloud

VSaaS Video Surveillance as a Service

VSS Video Surveillance System

Part VI

Bibliography

Bibliography

[1] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018. xix, 6, 17, 18, 19, 56, 57

[2] Gaurav Batra, Zach Jacobson, Siddarth Madhav, Andrea Queirolo, and Nick Santhanam. Artificial-intelligence hardware: New opportunities for semiconductor companies. *McKinsey and Company, January*, 2, 2019. xix, 17, 24, 25, 27

[3] Vijay Janapa Reddi, Christine Cheng, David Kanter, Peter Mattson, Guenther Schmuelling, Carole-Jean Wu, Brian Anderson, Maximilien Breughe, Mark Charlebois, William Chou, et al. Mlperf inference benchmark. In *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, pages 446–459. IEEE, 2020. xix, xx, 17, 26, 27, 37, 41, 46, 49, 67, 76

[4] Berk Ulker, Sander Stuijk, Henk Corporaal, and Rob G. J. Wijnhoven. Reviewing inference performance of state-of-the-art deep learning frameworks. *Proceedings of the 23th International Workshop on Software and Compilers for Embedded Systems*, 2020. xix, 30

[5] Carlo S. Regazzoni, Andrea Cavallaro, Ying Wu, Janusz Konrad, and Arun Hampapur. Video analytics for surveillance: Theory and practice [from the guest editors]. *IEEE Signal Processing Magazine*, 27:16–17, 2010. 3

[6] Guruh Fajar Shidik, Edi Noersasongko, Adhitya Nugraha, Pulung Nurtantio Andono, Jumanto Jumanto, and Edi Jaya Kusuma. A systematic review of intelligence video surveillance: trends, techniques, frameworks, and datasets. *IEEE Access*, 7:170457–170473, 2019. 3, 32

[7] Mark Treveil, Nicolas Omont, Clément Stenac, Kenji Lefevre, Du Phan, Joachim Zentici, Adrien Lavoillotte, Makoto Miyazaki, and Lynn Heidmann. *Introducing MLOps*. O'Reilly Media, 2020. 4

[8] Alexander Lavin, Ciarán M. Gilligan-Lee, Alessya Visnjic, Siddha Ganju, Dava Newman, Atilim Gunes Baydin, Sujoy Ganguly, Danny B. Lange, Ajay Sharma, Stephan Zheng, Eric P. Xing, Adam Gibson, James Parr, Chris Mattmann, and Yarin Gal. Technology readiness levels for machine learning systems. *Nature Communications*, 13, 2020. 4

[9] Tomasz Marcin Mucha, Sijia Ma, and Kaveh Abhari. Beyond mlops: The lifecycle of machine learning-based solutions. In *Americas Conference on Information Systems*, 2022. 4

[10] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *ArXiv*, abs/1710.09282, 2017. 6, 20

[11] Kai Han, Yunhe Wang, Hanting Chen, Xinghao Chen, Jianyuan Guo, Zhenhua Liu, Yehui Tang, An Xiao, Chunjing Xu, Yixing Xu, Zhaohui Yang, Yiman Zhang, and Dacheng Tao. A survey on vision transformer. *IEEE transactions on pattern analysis and machine intelligence*, PP, 2022. 6, 20

[12] Yanghao Li, Saining Xie, Xinlei Chen, Piotr Dollár, Kaiming He, and Ross B. Girshick. Benchmarking detection transfer learning with vision transformers. *ArXiv*, abs/2111.11429, 2021. 6, 20

[13] Albert Reuther, Peter Michaleas, Michael Jones, Vijay Gadepally, Siddharth Samsi, and Jeremy Kepner. Survey of machine learning accelerators. In *2020 IEEE high performance extreme computing conference (HPEC)*, pages 1–12. IEEE, 2020. 6, 25

[14] TensorflowLite. ML for mobile and edge devices. `https://www.tensorflow.org/lite`, 2023. [Online; accessed 19-FEB-2023]. 7, 29

[15] NVidia TensorRT. `https://developer.nvidia.com/tensorrt`, 2023. [Online; accessed 19-FEB-2023]. 7

[16] OpenVINO. . https://docs.openvino.ai/latest/home.html, 2023. [Online; accessed 19-FEB-2023]. 7

[17] Gaurav Verma, Yashi Gupta, Abid M Malik, and Barbara Chapman. Performance evaluation of deep learning compilers for edge inference. In *2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, pages 858–865. IEEE, 2021. 7

[18] Hanan Elazhary. Internet of things (iot), mobile cloud, cloudlet, mobile iot, iot cloud, fog, mobile edge, and edge emerging computing paradigms: Disambiguation and research directions. *Journal of Network and Computer Applications*, 128:105–140, 2019. 8

[19] T.Prem Jacob, A. Pravin, and R. Rajakumar. An ai-powered smart camera for object detection. In *2021 6th International Conference on Communication and Electronics Systems (ICCES)*, pages 1701–1703, 2021. 8

[20] Liangzhi Li, Kaoru Ota, and Mianxiong Dong. Deep learning for smart industry: Efficient manufacture inspection system with fog computing. *IEEE Transactions on Industrial Informatics*, 14(10):4665–4673, 2018. 8

[21] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. *Present and ulterior software engineering*, pages 195–216, 2017. 9

[22] Eric Chung, Jeremy Fowers, Kalin Ovtcharov, Michael Papamichael, Adrian Caulfield, Todd Massengill, Ming Liu, Daniel Lo, Shlomi Alkalay, Michael Haselman, Maleen Abeydeera, Logan Adams, Hari Angepat, Christian Boehn, Derek Chiou, Oren Firestein, Alessandro Forin, Kang Su Gatlin, Mahdi Ghandi, Stephen Heil, Kyle Holohan, Ahmad El Husseini, Tamas Juhasz, Kara Kagi, Ratna K. Kovvuri, Sitaram Lanka, Friedel van Megen, Dima Mukhortov, Prerak Patel, Brandon Perez, Amanda Rapsang, Steven Reinhardt, Bita Rouhani, Adam Sapek, Raja Seera, Sangeetha Shekar, Balaji Sridharan, Gabriel Weisz, Lisa Woods, Phillip Yi Xiao, Dan Zhang, Ritchie Zhao, and Doug Burger. Serving dnns in real time at datacenter scale with project brainwave. *IEEE Micro*, 38(2):8–20, 2018. 9

[23] Garrett McGrath and Paul R. Brenner. Serverless computing: Design, implementation, and performance. In *2017 IEEE 37th International Conference on Distributed Computing Systems Workshops (ICDCSW)*, pages 405–410, 2017. 9

[24] Unai Elordi, Luis Unzueta, Ignacio Arganda-Carreras, and Oihana Otaegui. How can deep neural networks be generated efficiently for devices with limited resources? In *Articulated Motion and Deformable Objects - 10th International Conference, AMDO 2018, Palma de Mallorca, Spain, July 12-13, 2018, Proceedings*, pages 24–33, 2018. 11, 17

[25] Unai Elordi, Luis Unzueta, Jon Goenetxea, Sergio Sanchez-Carballido, Ignacio Arganda-Carreras, and Oihana Otaegui. Benchmarking deep neural network inference performance on serverless environments with mlperf. *IEEE Software*, 38(1):81–87, 2021. 11

[26] Unai Elordi, Luis Unzueta, Jon Goenetxea, Estíbaliz Loyo, Ignacio Arganda-Carreras, and Oihana Otaegui. On-demand serverless video surveillance with optimal deployment of deep neural networks. In *Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP 2021, Volume 4: VISAPP, Online Streaming, February 8-10, 2021*, pages 717–723, 2021. 11

[27] Unai Elordi, Álvaro Bertelsen, Luis Unzueta, Nerea Aranjuelo, Jon Goenetxea, and Ignacio Arganda-Carreras. Optimal deployment of face recognition solutions in a heterogeneous iot platform for secure elderly care applications. In Jaroslaw Watróbski, Wojciech Salabun, Carlos Toro, Cecilia Zanni-Merk, Robert J. Howlett, and Lakhmi C. Jain, editors, *Knowledge-Based and Intelligent Information & Engineering Systems: Proceedings of the 25th International Conference KES-2021, Virtual Event / Szczecin, Poland, 8-10 September 2021*, volume 192 of *Procedia Computer Science*, pages 3204–3213. Elsevier, 2021. 11, 79, 80

[28] Unai Elordi, Chiara Lunerti, Luis Unzueta, Jon Goenetxea, Nerea Aranjuelo, Alvaro Bertelsen, and Ignacio Arganda-Carreras. Designing automated deployment strategies of face recognition solutions in heterogeneous iot platforms. *Inf.*, 12(12):532, 2021. 11

[29] Andrea Macarulla Rodriguez, Luis Unzueta, Zeno Geradts, Marcel Worring, and U. Elordi. Multi-task explainable quality networks for large-scale forensic facial recognition. *IEEE Journal of Selected Topics in Signal Processing*, 2023. 11, 70

[30] Luis Unzueta, Sandra Garcia, Jorge García, Valentin Corbin, Nerea Aranjuelo, U. Elordi, Oihana Otaegui, and Maxime Danielli. Building a camera-based smart sensing system for digitalized on-demand aircraft cabin readiness verification. In *ROBOVIS*, 2020. 11

[31] Paul C. Castro, Vatche Ishakian, Vinod Muthusamy, and Aleksander Slominski. The server is dead, long live the server: Rise of serverless computing, overview of current state and future trends in research and industry. *CoRR*, abs/1906.02888, 2019. 17, 33, 42

[32] Forrest N Iandola, Song Han, Matthew W Moskewicz, Khalid Ashraf, William J Dally, and Kurt Keutzer. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and< 0.5 mb model size. *arXiv preprint arXiv:1602.07360*, 2016. 18, 19, 23

[33] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. 18, 19, 80

[34] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *CoRR*, abs/1707.01083, 2017. 18, 19

[35] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4510–4520, 2018. 18, 20

[36] Seyyed Hossein Hasanpour, Mohammad Rouhani, Mohsen Fayyaz, Mohammad Sabokrou, and Ehsan Adeli. Towards principled design of deep convolutional networks: Introducing simpnet. *arXiv preprint arXiv:1802.06205*, 2018. 18, 20

[37] 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. 18, 21, 28

[38] Shuchang Zhou, Yuxin Wu, Zekun Ni, Xinyu Zhou, He Wen, and Yuheng Zou. Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients. *arXiv preprint arXiv:1606.06160*, 2016. 18, 21

[39] Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in neural information processing systems*, pages 3123–3131, 2015. 18, 21

[40] 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: Learning*, 2016. 18, 21

[41] 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. 18, 21

[42] Felix Juefei-Xu, Vishnu Naresh Boddeti, and Marios Savvides. Local binary convolutional neural networks. *CoRR*, abs/1608.06049, 2016. 18, 21

[43] 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. 18, 21, 23, 56

[44] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. pages 598–605, 1990. 18, 22

[45] Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993. 18, 22

[46] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 18, 23

[47] Martin Abadi et al. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*, 2016. 18, 23

[48] Huizi Mao, Song Han, Jeff Pool, Wenshuo Li, Xingyu Liu, Yu Wang, and William J Dally. Exploring the regularity of sparse structure in convolutional neural networks. *arXiv preprint arXiv:1705.08922*, 2017. 18, 23

[49] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *CoRR*, abs/1512.08571, 2015. 18

[50] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. 18, 23

[51] 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. 18, 23, 80

[52] Jian-Hao Luo, Jianxin Wu, and Weiyao Lin. Thinet: A filter level pruning method for deep neural network compression. *2017 IEEE International Conference on Computer Vision (ICCV)*, pages 5068–5076, 2017. 18, 23

[53] Babajide O Ayinde and Jacek M Zurada. Building efficient convnets using redundant feature pruning. *arXiv preprint arXiv:1802.07653*, 2018. 18, 23

[54] Jeng-Hau Lin, Tianwei Xing, Ritchie Zhao, Zhiru Zhang, Mani Srivastava, Zhuowen Tu, and Rajesh K. Gupta. Binarized convolutional neural networks with separable filters for efficient hardware acceleration. In *2017 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pages 344–352, 2017. 18, 24

[55] Qibin Zhao, Guoxu Zhou, Shengli Xie, Liqing Zhang, and Andrzej Cichocki. Tensor ring decomposition. *CoRR*, abs/1606.05535, 2016. 18, 24

[56] Wenqi Wang, Yifan Sun, Brian Eriksson, Wenlin Wang, and Vaneet Aggarwal. Wide compression: Tensor ring nets. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pages 9329–9338. Computer Vision Foundation / IEEE Computer Society, 2018. 18, 24

[57] Matthew Sotoudeh and Sara S Baghsorkhi. Deepthin: A self-compressing library for deep neural networks. *arXiv preprint arXiv:1802.06944*, 2018. 18, 24

[58] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 18, 19

[59] Wei Liu, Dragomir Anguelov, Dumitru Erhan, Christian Szegedy, Scott Reed, Cheng-Yang Fu, and Alexander C. Berg. Ssd: Single shot multibox detector. In Bastian Leibe, Jiri Matas, Nicu Sebe, and Max Welling, editors, *Computer Vision – ECCV 2016*, pages 21–37, Cham, 2016. Springer International Publishing. 18, 19, 80

[60] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 248–255, 2009. 18, 52, 95, 105

[61] Gao Huang, Zhuang Liu, Laurens van der Maaten, and Kilian Q. Weinberger. Densely connected convolutional networks. In *IEEE Conf. on Computer Vision and Pattern Recognition, CVPR, Honolulu, HI, USA*, pages 2261–2269, 2017. 19

[62] Andrew G. Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, Quoc V. Le, and Hartwig Adam. Searching for mobilenetv3. *2019 IEEE/CVF International Conference on Computer Vision (ICCV)*, pages 1314–1324, 2019. 20

[63] Mingxing Tan and Quoc V. Le. Efficientnet: Rethinking model scaling for convolutional neural networks. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6105–6114. PMLR, 2019. 20

[64] Mingxing Tan and Quoc V. Le. Efficientnetv2: Smaller models and faster training. In *Proc. Int. Conf. on Machine Learning, ICML, 18-24 July, Virtual Event*, volume 139 of *Proc. of Machine Learning Research*, pages 10096–10106, 2021. 20, 98, 105, 106, 109

[65] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. 2018. 23

[66] Sajid Anwar, Kyuyeon Hwang, and Wonyong Sung. Structured pruning of deep convolutional neural networks. *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, 13(3):32, 2017. 23

[67] Yang He, Xuanyi Dong, Guoliang Kang, Yanwei Fu, and Yi Yang. Progressive deep neural networks acceleration via soft filter pruning. *ArXiv*, abs/1808.07471, 2018. 23

[68] Zhuang Liu, Mingjie Sun, Tinghui Zhou, Gao Huang, and Trevor Darrell. Rethinking the value of network pruning. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. 23, 56

[69] Manar AbuTalib, Sohaib Majzoub, Qassim Nasir, and Dina J. Hejji. A systematic literature review on hardware implementation of artificial intelligence algorithms. *J. Supercomput.*, 77(2):1897–1938, 2021. 26

[70] Norman P. Jouppi, Cliff Young, Nishant Patil, David A. Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, Rick Boyle, Pierre-luc Cantin, Clifford Chao, Chris Clark, Jeremy Coriell, Mike Daley, Matt Dau, Jeffrey Dean, Ben Gelb, Tara Vazir Ghaemmaghami, Rajendra Gottipati, William Gulland, Robert Hagmann, C. Richard Ho, Doug Hogberg, John Hu, Robert Hundt, Dan Hurt, Julian Ibarz, Aaron Jaffey, Alek Jaworski, Alexander Kaplan, Harshit Khaitan, Daniel Killebrew, Andy Koch, Naveen Kumar, Steve Lacy, James Laudon, James Law, Diemthu Le, Chris Leary, Zhuyuan Liu, Kyle Lucke, Alan Lundin, Gordon MacKean, Adriana Maggiore, Maire Mahony, Kieran Miller, Rahul Nagarajan, Ravi Narayanaswami, Ray Ni, Kathy Nix, Thomas Norrie, Mark Omernick, Narayana Penukonda, Andy Phelps, Jonathan Ross, Matt Ross, Amir Salek, Emad

Samadiani, Chris Severn, Gregory Sizikov, Matthew Snelham, Jed Souter, Dan Steinberg, Andy Swing, Mercedes Tan, Gregory Thorson, Bo Tian, Horia Toma, Erick Tuttle, Vijay Vasudevan, Richard Walter, Walter Wang, Eric Wilcox, and Doe Hyun Yoon. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture, ISCA 2017, Toronto, ON, Canada, June 24-28, 2017*, pages 1–12. ACM, 2017. 27

[71] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, 2017. 27

[72] Han Vanholder. Efficient inference with tensorrt. In *GPU Technology Conference*, volume 1, page 2, 2016. 29

[73] Yury Gorbachev, Mikhail Fedorov, Iliya Slavutin, Artyom Tugarev, Marat Fatekhov, and Yaroslav Tarkan. Openvino deep learning workbench: Comprehensive analysis and tuning of neural networks inference. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pages 783–787. IEEE, 2019. 29, 31

[74] ONNX Runtime. home. `https://onnxruntime.ai/`, 2023. [Online; accessed 26-APR-2023]. 29

[75] Omais Shafi, Chinmay Rai, Rijurekha Sen, and Gayathri Ananthanarayanan. Demystifying tensorrt: Characterizing neural network inference engine on nvidia edge devices. *2021 IEEE International Symposium on Workload Characterization (IISWC)*, pages 226–237, 2021. 30

[76] Mingzhen Li, Yi Liu, Xiaoyan Liu, Qingxiao Sun, Xin You, Hailong Yang, Zhongzhi Luan, and Depei Qian. The deep learning compiler: A comprehensive survey. *IEEE Transactions on Parallel and Distributed Systems*, 32:708–727, 2020. 30

[77] Alexander Demidovskij, Artyom Tugaryov, Alexander Suvorov, Yaroslav Tarkan, Marat Fatekhov, Igor Salnikov, Andrey Kashchikhin, Vladimir Golubenko, Galina Dedyukhina, Alina Alborova, Ryan Palmer, Mikhail Fedorov, and Yury Gorbachev.

Openvino deep learning workbench: A platform for model optimization, analysis and deployment. In *2020 IEEE 32nd International Conference on Tools with Artificial Intelligence (ICTAI)*, pages 661–668, 2020. 31

[78] Thanathip Limna and Pichaya Tandayya. A flexible and scalable component-based system architecture for video surveillance as a service, running on infrastructure as a service. *Multim. Tools Appl.*, 75(4):1765–1791, 2016. 32, 54

[79] Chengliang Zhang, Minchen Yu, Wei Wang, and Feng Yan. Mark: Exploiting cloud services for cost-effective, slo-aware machine learning inference serving. In Dahlia Malkhi and Dan Tsafrir, editors, *2019 USENIX Annual Technical Conference, USENIX ATC 2019, Renton, WA, USA, July 10-12, 2019*, pages 1049–1062. USENIX Association, 2019. 33

[80] Francisco Romero, Qian Li, Neeraja J. Yadwadkar, and Christos Kozyrakis. Infaas: Automated model-less inference serving. In Irina Calciu and Geoff Kuenning, editors, *2021 USENIX Annual Technical Conference, USENIX ATC 2021, July 14-16, 2021*, pages 397–411. USENIX Association, 2021. 33

[81] DeepStream SDK. DeepStream SDK | NVIDIA Developers. `https://onnxruntime.ai/`, 2023. [Online; accessed 26-APR-2023]. 34

[82] Intel Deep Learning streamer. `https://docs.openvino.ai/latest/openvino_docs_dlstreamer.html`, 2023. [Online; accessed 28-APR-2023]. 34

[83] Yecheng Xiang and Hyoseung Kim. Pipelined data-parallel CPU/GPU scheduling for multi-dnn real-time inference. *IEEE Real-Time Systems Symposium (RTSS)*, pages 392–405, 2019. 34, 108

[84] CheolSu Lim and Myungsun Kim. ODMDEF: On-device multi-DNN execution framework utilizing adaptive layer-allocation on general purpose cores and accelerators. *IEEE Access*, 9:85403–85417, 2021. 34, 108

[85] Eunji Jeong, Jangryul Kim, Samnieng Tan, Jaeseong Lee, and Soonhoi Ha. Deep learning inference parallelization on heterogeneous processors with tensorRT. *IEEE Embedded Systems Letters*, pages 1–1, 2021. 34, 108

[86] Eunjin Jeong, Jangryul Kim, and Soonhoi Ha. TensorRT-based framework and optimization methodology for deep learning inference on Jetson boards. *ACM Trans. Embed. Comput. Syst.*, 21(5):51:1–51:26, 2022. 35, 108, 109

[87] Myungsun Kim. Guaranteeing that multi-level prioritized DNN models on an embedded GPU have inference performance proportional to respective priorities. *IEEE Embedded Systems Letters*, 2021. 35, 108

[88] Bart Cox, Jeroen Galjaard, Amirmasoud Ghiassi, Robert Birke, and Lydia Yiyu Chen. MASA: Responsive multi-DNN inference on the edge. *IEEE International Conference on Pervasive Computing and Communications (PerCom)*, pages 1–10, 2021. 35, 108

[89] Fuxun Yu, Shawn Bray, Di Wang, Longfei Shangguan, Xulong Tang, Chenchen Liu, and Xiang Chen. Automated runtime-aware scheduling for multi-tenant DNN inference on GPU. *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–9, 2021. 35, 108

[90] Zhuoqing Chang, Shubo Liu, Xingxing Xiong, Zhaohui Cai, and Guoqing Tu. A survey of recent advances in edge-computing-powered artificial intelligence of things. *IEEE Internet of Things Journal*, 8:13849–13875, 2021. 35

[91] Ashkan Yousefpour, Caleb Fung, Tam Nguyen, Krishna Kadiyala, Fatemeh Jalali, Amirreza Niakanlahiji, Jian Kong, and Jason P. Jue. All one needs to know about fog computing and related edge computing paradigms: A complete survey. *J. Syst. Archit.*, 98:289–330, 2019. 36

[92] Pengfei Hu, Huansheng Ning, Tie Qiu, Yanfei Zhang, and Xiong Luo. Fog computing based face identification and resolution scheme in internet of things. *IEEE Trans. Ind. Informatics*, 13(4):1910–1920, 2017. 36, 80

[93] Pengfei Hu, Huansheng Ning, Tie Qiu, Houbing Song, Yanna Wang, and Xuanxia Yao. Security and privacy preservation scheme of face identification and resolution framework using fog computing in internet of things. *IEEE Internet Things J.*, 4(5):1143–1155, 2017. 36, 80

[94] Yitu Wang and Takayuki Nakachi. A privacy-preserving learning framework for face recognition in edge and cloud networks. *IEEE Access*, 8:136056–136070, 2020. 36, 80

[95] A privacy-preserving deep learning approach for face recognition with edge computing. In Irfan Ahmad and Swaminathan Sundararaman, editors, *USENIX Workshop on Hot Topics in Edge Computing, HotEdge 2018, Boston, MA, July 10, 2018*. USENIX Association, 2018. 37, 80

[96] Daniel Bardsley, Larry Ryan, and John Howard. Serverless performance and optimization strategies. In *2018 IEEE International Conference on Smart Cloud, SmartCloud 2018, New York City, NY, USA, September 21-23, 2018*, pages 19–26. IEEE, 2018. 42, 44

[97] Pascal Maissen, Pascal Felber, Peter G. Kropf, and Valerio Schiavoni. Faasdom: a benchmark suite for serverless computing. In Julien Gascon-Samson, Kaiwen Zhang, Khuzaima Daudjee, and Bettina Kemme, editors, *14th ACM International Conference on Distributed and Event-based Systems, DEBS 2020, Montreal, Quebec, Canada, July 13-17, 2020*, pages 73–84. ACM, 2020. 43, 52

[98] Tae Joon Jun, Daeyoun Kang, Dohyeun Kim, and Daeyoung Kim. Gpu enabled serverless computing framework. In Ivan Merelli, Pietro Liò, and Igor V. Kotenko, editors, *26th Euromicro International Conference on Parallel, Distributed and Network-based Processing, PDP 2018, Cambridge, United Kingdom, March 21-23, 2018*, pages 533–540. IEEE Computer Society, 2018. 43

[99] A Baird, G Huang, C Munns, and O Weinstein. Serverless architectures with aws lambda: Overview and best practices, 2017. 44

[100] Josef Spillner, Cristian Mateos, and David A. Monge. Faaster, better, cheaper: The prospect of serverless scientific computing and HPC. In Esteban E. Mocskos and Sergio Nesmachnow, editors, *High Performance Computing - 4th Latin American Conference, CARLA 2017, Buenos Aires, Argentina, and Colonia del Sacramento, Uruguay, September 20-22, 2017, Revised Selected Papers*, volume 796 of *Communications in Computer and Information Science*, pages 154–168. Springer, 2017. 48, 56

[101] Ioana Baldini, Paul C. Castro, Kerry Shih-Ping Chang, Perry Cheng, Stephen Fink, Vatche Ishakian, Nick Mitchell, Vinod Muthusamy, Rodric Rabbah, Aleksander Slominski, and Philippe Suter. Serverless computing: Current trends and open problems. In Sanjay Chaudhary, Gaurav Somani, and Rajkumar Buyya, editors, *Research Advances in Cloud Computing*, pages 1–20. Springer, 2017. 50

[102] M. Garrett McGrath and Paul R. Brenner. Serverless computing: Design, implementation, and performance. In Aibek Musaev, João Eduardo Ferreira, and Teruo Higashino, editors, *37th IEEE International Conference on Distributed Computing Systems Workshops, ICDCS Workshops 2017, Atlanta, GA, USA, June 5-8, 2017*, pages 405–410. IEEE Computer Society, 2017. 50

[103] Intel. OpenCV:Open Source Computer Vision Library. `https://opencv.org`, 2023. [Online; accessed 19-FEB-2023]. 51

[104] Intel. OpenVINO:open-source toolkit for optimizing and deploying deep learning models. `https://github.com/openvinotoolkit/openvino`, 2023. [Online; accessed 19-FEB-2023]. 51

[105] Tsung-Yi Lin, Michael Maire, Serge J. Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft COCO: common objects in context. In *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, volume 8693 of *Lecture Notes in Computer Science*, pages 740–755, 2014. 52, 95

[106] Luciano Baresi, Danilo Filgueira Mendonça, Martin Garriga, Sam Guinea, and Giovanni Quattrocchi. A unified model for the mobile-edge-cloud continuum. *ACM Trans. Internet Techn.*, 19(2):29:1–29:21, 2019. 54, 124

[107] Sathyan Munirathinam. Chapter six - industry 4.0: Industrial internet of things (IIOT). *Adv. Comput.*, 117:129–164, 2020. 66

[108] Muhtahir O. Oloyede, Gerhard P. Hancke, and Hermanus Carel Myburgh. A review on face recognition systems: recent approaches and challenges. *Multim. Tools Appl.*, 79(37-38):27891–27922, 2020. 66

[109] Murat Taskiran, Nihan Kahraman, and Cigdem Eroglu Erdem. Face recognition: Past, present and future (a review). *Digit. Signal Process.*, 106:102809, 2020. 66

[110] Yassin Kortli, Maher Jridi, Ayman Al Falou, and Mohamed Atri. Face recognition systems: A survey. *Sensors*, 20(2), 2020. 66

[111] Anil K Jain, Patrick Flynn, and Arun A Ross. *Handbook of biometrics*. Springer Science & Business Media, 2007. 67

[112] Lidia Bajenaru, Ion Alexandru Marinescu, Ciprian Dobre, Gabriel Ioan Prada, and Costas S. Constantinou. Towards the development of a personalized healthcare solution for elderly: from user needs to system specifications. In *12th International Conference on Electronics, Computers and Artificial Intelligence, ECAI 2020, Bucharest, Romania, June 25-27, 2020*, pages 1–6. IEEE, 2020. 67

[113] Ramon Blanco-Gonzalo, Chiara Lunerti, Raul Sanchez-Reillo, and Richard Michael Guest. Biometrics: Accessibility challenge or opportunity? *PLOS ONE*, 13(3):1–20, 03 2018. 68

[114] Valentin Bazarevsky, Yury Kartynnik, Andrey Vakunov, Karthik Raveendran, and Matthias Grundmann. Blazeface: Sub-millisecond neural face detection on mobile gpus. *CoRR*, abs/1907.05047, 2019. 69

[115] Yury Kartynnik, Artsiom Ablavatski, Ivan Grishchenko, and Matthias Grundmann. Real-time facial surface geometry from monocular video on mobile gpus. In *CVPR Workshop on Computer Vision for Augmented and Virtual Reality 2019*, Long Beach, CA, 2019. 69, 72

[116] Torsten Schlett, Christian Rathgeb, Olaf Henniger, Javier Galbally, Julian Fierrez, and Christoph Busch. Face image quality assessment: A literature survey. *ACM Comput. Surv.*, dec 2021. 70

[117] Heinz Hofbauer, Luca Debiasi, Susanne Kränkl, and Andreas Uhl. Exploring presentation attack vulnerability and usability of face recognition systems. *IET Biom.*, 10(2):219–232, 2021. 70

[118] Kavita Kavita, Gurjit Singh Walia, and Rajesh Rohilla. A contemporary survey of unimodal liveness detection techniques: Challenges amp; opportunities. In *2020 3rd International Conference on Intelligent Sustainable Systems (ICISS)*, pages 848–855, 2020. 70

[119] Yuanhan Zhang, ZhenFei Yin, Yidong Li, Guojun Yin, Junjie Yan, Jing Shao, and Ziwei Liu. Celeba-spoof: Large-scale face anti-spoofing dataset with rich annotations. In Andrea Vedaldi, Horst Bischof, Thomas Brox, and Jan-Michael Frahm, editors, *Computer Vision – ECCV 2020*, pages 70–85, Cham, 2020. Springer International Publishing. 70

[120] M. Swapna, Yogesh Kumar Sharma, and B. M. G. Prasad. A survey on face recognition using convolutional neural network. In K. Srujan Raju, Roman Senkerik, Satya Prasad Lanka, and V. Rajagopal, editors, *Data Engineering and Communication Technology*, pages 649–661, Singapore, 2020. Springer Singapore. 70

[121] Hyoukjun Kwon, Liangzhen Lai, Michael Pellauer, Tushar Krishna, Yu-Hsin Chen, and Vikas Chandra. Heterogeneous dataflow accelerators for multi-dnn workloads. In *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, pages 71–83, 2021. 75

[122] Guangcan Mai, Kai Cao, Pong C. Yuen, and Anil K. Jain. On the reconstruction of face images from deep face templates. *IEEE Trans. Pattern Anal. Mach. Intell.*, 41(5):1188–1202, 2019. 77

[123] Vishnu Naresh Boddeti. Secure face matching using fully homomorphic encryption. In *9th IEEE International Conference on Biometrics Theory, Applications and Systems, BTAS 2018, Redondo Beach, CA, USA, October 22-25, 2018*, pages 1–10. IEEE, 2018. 78

[124] Dorottya Papp, Máté Zsombor, and Levente Buttyán. Tee-based protection of cryptographic keys on embedded iot devices. In *Annales Mathematicae et Informaticae*, volume 53, pages 245–256. Eszterházy Károly Egyetem Líceum Kiadó, 2021. 78

[125] Mihály Héder. From nasa to eu: the evolution of the trl scale in public sector innovation. *The Innovation Journal*, 22:1, 2017. 92

[126] Zhong-Qiu Zhao, Peng Zheng, Shou tao Xu, and Xindong Wu. Object detection with deep learning: A review. *IEEE Transactions on Neural Networks and Learning Systems*, 30:3212–3232, 2018. 93

[127] Yiran Chen, Yuan Xie, Linghao Song, Fan Chen, and Tianqi Tang. A survey of accelerator architectures for deep neural networks. *Engineering*, 6:264–274, 2020. 93

[128] Li Liu, Wanli Ouyang, Xiaogang Wang, Paul W. Fieguth, Jie Chen, Xinwang Liu, and Matti Pietikäinen. Deep learning for generic object detection: A survey. *International Journal of Computer Vision*, 128:261–318, 2019. 93

[129] Wei Wang, Yujing Yang, Xin Wang, Weizheng Wang, and Ji Li. Development of convolutional neural network and its application in image classification: a survey. *Optical Engineering*, 58:040901 – 040901, 2019. 93

[130] Yu Zhang and Qiang Yang. A survey on multi-task learning. *IEEE Trans. on Knowledge and Data Engineering*, 2021. 95, 96

[131] Fang Wang, Hu Han, Shiguang Shan, and Xilin Chen. Deep multi-task learning for joint prediction of heterogeneous face attributes. In *2017 12th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2017)*, pages 173–179. IEEE, 2017. 95, 106

[132] Vivien Sainte Fare Garnot and Loic Landrieu. Leveraging class hierarchies with metric-guided prototype learning. In *The British Machine Vision Conference, BMVC, 22-25 November, Virtual Event*, 2021. 95, 97, 98, 106

[133] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper R. R. Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Alexander Kolesnikov, Tom Duerig, and Vittorio Ferrari. The open images dataset V4. *Int. J. Comput. Vis.*, 128(7):1956–1981, 2020. 95

[134] Unai Elordi, Nerea Aranjuelo, Luis Unzueta, Jose Luis Apellaniz, Jorge García, and Oihana Otaegui. SmaCS dataset, January 2023. 95, 96, 102, 103, 116, 123

[135] Mei Wang and Weihong Deng. Deep visual domain adaptation: A survey. *Neurocomputing*, 312:135–153, 2018. 96, 97

[136] Nerea Aranjuelo, Jorge García, Luis Unzueta, Sara García, Unai Elordi, and Oihana Otaegui. Building synthetic simulated environments for configuring and training multi-camera systems for surveillance applications. In *Proc. Int. Joint Conf. on Computer Vision, Imaging and Computer Graphics Theory and Applications, VISIGRAPP, Volume 5: VISAPP, Online Streaming, February 8-10*, pages 80–91, 2021. 97, 102

[137] Nerea Aranjuelo, Jose Luis Apellaniz, Luis Unzueta, Jorge García, Sara García, Unai Elordi, and Oihana Otaegui. Leveraging synthetic data for dnn-based visual analysis of passenger seats. *SN Computer Science*, [in press]. 97, 106

[138] Jake Snell, Kevin Swersky, and Richard S. Zemel. Prototypical networks for few-shot learning. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems, December 4-9, 2017, Long Beach, CA, USA*, pages 4077–4087, 2017. 99

[139] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. Yolov4: Optimal speed and accuracy of object detection. *CoRR*, abs/2004.10934, 2020. 109, 110

[140] Jetson-aware embedded deep learning inference acceleration framework with TensorRT. `https://github.com/cap-lab/jedi`. Accessed: 2023-05-29. 109, 110, 116

[141] Gastão Florêncio Miranda Jr., Carlos Eduardo Thomaz, and Gilson Antonio Giraldi. Geometric data analysis based on manifold learning with applications for image understanding. In *30th SIBGRAPI Conference on Graphics, Patterns and Images - Tutorials, SIBGRAPI-T 2017, Niterói, Brazil, October 17-18, 2017*, pages 42–62. IEEE Computer Society, 2017. 112