Optimizing Edge Cloud Deployments for Video Analytics

By Daniel Rivas

Dissertation submitted in partial fulfillment of the requirements for the

Doctoral Degree in Computer Architecture

Universitat Politècnica de Catalunya - BarcelonaTECH

July 2022

Advisors:
Dr. Francesc Guim
Dr. David Carrera
Tutor:
Dr. Yolanda Becerra
Affiliation:
Department of Computer Architecture
Universitat Politècnica de Catalunya - BarcelonaTECH
Location
Barcelona, Spain

Abstract

As our digital world and physical realities blend together, we, as users, are growing to expect real-time interaction wherever and whenever we want. Newer internet services require lower latency than a data center hundreds of kilometers away can provide, while generating more data than the backhaul of the network can absorb. Consequently, resources are being moved to the edge of the network to create a new type of highly distributed cloud infrastructure: *the edge cloud*.

The edge cloud brings compute and storage close to the end-user. That is, close to where data is produced. Thus, service providers have a first layer of resources at their disposal, with which to process and filter upstream information to accelerate access to their services and optimize the amount of information sent to the public cloud.

Edge cloud deployments differ from traditional cloud deployments in two crucial points that, altogether, arise new challenges. First, locations are constrained by the amount of resources they can host, limiting the services these locations can execute and provide. Moreover, this limitation causes service providers to have a narrower spectrum of hardware from which to choose when executing their services at a given location while increasing the heterogeneity of the infrastructure as a whole. That is, resource-constrained nodes and hardware accelerated server-grade nodes will coexist, not within the same location, but within the same network. Services must consider this heterogeneity and adapt to it transparently. Second, contrary to traditional cloud locations, user aggregation is limited in the edge, as each location serves only its designated geographical region. Consequently, service providers face the challenge of understanding the impact of considering locality, hardware heterogeneity, and the compute requirements of their workloads while designing new deployments.

Among all the services and use cases expected to be accelerated from the edge, one stands out from the rest. Video analytics is already today the main use case being deployed to the edge due to its strict latency requirements and the high amount of bandwidth it continuously generates. In fact, the edge cloud is deemed as a necessary accelerator for video analytics deployments to be feasible and cost-effective at scale. Unfortunately, video analytics is a computationally expensive task, often requiring high-end hardware acceleration to provide real-time performance to a single user. In a context in which resources become scarcer the closer they are to the edge of the network, such high computational cost easily exceeds what the infrastructure can provide unless the entire workflow is optimized. Therefore, there is a need for new techniques that can enable video analytics to be served from resource-constrained nodes without compromising the user experience, while, at the same time, be able to maximize resource utilization there where resources are scarcer.

In this context, this thesis contributes to the optimization of edge cloud deployments aimed at providing service of video analytics workloads through distributed and resource-constrained edge infrastructures. Towards this end, this thesis contributes to the state-of-the-art by (C1) characterizing video analytics workloads on an heterogeneous set of hardware platform, each mapping to a different edge location archetype; (C2) presenting a novel framework to accelerate large-scale deployments by leveraging the potential of a hybrid Edge-Cloud interplay and automating the task of optimizing neural networks and specializing them to the specific deployment's context; and, finally, (C3) proposing a novel method that makes it possible for video analytics workloads to be massively distributed across different edge of locations.

Together, the contributions of this thesis define the hardware requirements of a heterogeneous edge cloud (C1) and open the door to new ways to adapt, optimize (C2), and distribute (C3) video analytics workloads for the edge cloud.

Acknowledgments

A doctoral dissertation is a journey, both personal and academic, that must be completed by oneself. However, this does not mean that the path has to be taken alone. On the contrary, research is a shared path that must be taken in company, sometimes even in the company of those we least expect.

On a more personal level, I would like to begin by thanking those who have been the two pillars on which much of this thesis has been based. On the one hand, my partner, Anna Wohlrab, whom, very optimistic me, I told when I first met her that my thesis would be ready in just a few months. Even years later, she never stopped supporting me, encouraging me, and putting up with my tantrums every time I saw the path as uncertain. On the other hand, I want to thank my two advisors, Dr. Francesc Guim and Dr. David Carrera. Cesc, as a colleague and friend in equal parts, was the one who back in 2017 told me ”Why don't you enroll for a PhD?” when it had never crossed my mind. That ended up being one of the decisions that would most impact my life from that moment on. David, on the other hand, always had an innate gift for redirecting me and all my ideas together in half an hour when I was unable to see the light at the end of the tunnel and even had already prepared the resignation speech. Thank you all, for never letting me quit.

In addition, I would like to make a special mention to a long list of people who have made it possible for me to make it to the end, as a person and a researcher. Thanks to Dr. Jordi Cortadella, who saw something in me, welcomed me as a student in his interesting projects where I did not stop learning and having fun and, as if that were not enough, introduced me to Cesc. Thanks to Dr. Jordà Polo, who acted as director, to Dr. Josep Ll. Berral and, in general, to the whole BSC-DC team for always being willing to help as a research team. Thanks to Yolanda Becerra for taking on the role tutor. Thanks to Dr. Daniel Domingo for teaching me that scientific progress can appear even in the most unlikely collaborations. Thanks to Dr. Vicent Costa for transmitting part of his immaculate scientific rigor to me part. Last but not least, thanks to my parents who, although they did not quite understand why I was returning to the university, always trusted that the right path would be whichever one I chose and never doubted that I would go all the way.

Agraïments

Una tesi doctoral és un camí, tant personal com acadèmic, que ha de fer un per si mateix. Tanmateix, això no significa que el camí es faci sol. Al contrari, la recerca és un camí que ha de fer-se acompanyat, a vegades fins i tot per les persones que menys esperem.

A nivell més personal m'agradaria començar agraint als que han estat els dos pilars sobre els quals s'ha sustentat gran part d'aquesta tesi. D'una banda, la meva parella, Anna Wohlrab, a qui, molt optimista jo, li vaig dir quan la vaig conèixer que estaria llest en tot just uns mesos. Encara anys després, mai va deixar de donar-me suport, d'animar-me ni d'aguantar els meus tantrums cada vegada que veia el camí incert. D'altra banda, els meus dos directors, el Dr. Francesc Guim i el Dr. David Carrera. Cesc, com a company i amic a parts iguals, va ser qui allà per 2017 em va dir "Per què no registres la tesi?" quan mai m'havia passat pel cap. Aquesta va acabar sent una de les decisions que més impactarien la meva vida a partir d'aquest moment. David, per part seva, sempre va tenir un do innat per a redirigir-me les totes idees juntes en mitja hora quan jo no veia la llum al final del túnel i ja tenia preparat fins al discurs de renúncia. Gràcies per mai deixar-me abandonar.

A més, m'agradaria fer esment especial a una llarga llista de persones que han fet possible que jo, com a persona i investigador, hagi aconseguit arribar fins al final. Gràcies al Dr. Jordi Cortadella, qui va veure alguna cosa en mi, em va acollir com a alumne en els seus interessants projectes on no vaig deixar d'aprendre i divertir-me i, per si no n'hi hagués prou, em va presentar a Cesc. Gràcies al Dr. Jordà Polo, que va fer sovint de director, al Dr. Josep Ll. Berral i, en general, a tot l'equip de BSC-DC per estar sempre disposat ajudar com a equip de recerca. Gràcies a Yolanda Becerra per prendre el rol de tutora. Gràcies al Dr. Daniel Domingo per ensenyar-me que l'avanç científic pot aparèixer fins i tot en les col·laboracions més rocambolesques. Gràcies al Dr. Vicent Costa per transmetre'm part del seu immaculat rigor científic. Finalment, i no per això menys important, gràcies als meus pares que, encara que no entenien molt bé per què tornava a la universitat, sempre van confiar que el camí correcte seria qualsevol que jo triés i mai van dubtar que arribaria fins al final.

Roads? Where we're going we don't need roads.

— Dr. Emmett Brown

Contents

List of Figures

List of Tables

Acronyms

ADAS	Advanced Driver-Assistance Systems
AR	Augmented Reality
CDN	Content Delivery Network
RAPL	Running Average Power Limit
TPU	Tensor Processor Unit
TSP	Telecommunication Service Provider
VNNI	Vector Neural Network Instructions
VR	Virtual Reality
VOD	Video On Demand

Chapter 1

Introduction

The internet, as we know it today, shapes our way of life through a myriad of services that are essential to the average person of the digital era. Every single one of these services relies on an ever-growing swarm of computers. Together, these computers guarantee 24/7 availability from anywhere in the world. This swarm is more commonly known as *the cloud*.

Over the last two decades, the cloud and cloud computing have paved the way for a new industrial revolution by removing the entry barrier for businesses to transition into modern computer systems. With it, a new digital revolution began, leading to an unprecedented growth in demand for compute, storage, and network resources. For years, cloud infrastructures were able to cope with the growing demand by pretty much stacking more resources in a handful of locations per geographical region; an ability granted by the level of user aggregation only such locations enjoy.

However, the *state of affairs* is pushing traditional cloud infrastructures to their limit. On one side, the increasing number of services pouring data non-stop risk the saturation of the backhaul of the network. On the other side, emerging use cases, driven by the promise of sub-millisecond latencies with 5G networks, assume such low latency to work that accessing central locations hundreds of kilometers away is out of the picture unless some physical laws are broken down the line. Consequently, the cloud, as we know it, must evolve.

A new form of cloud is emerging to overcome these limitations: the edge cloud. The edge cloud moves resources to the edge of the network, closer to the user. Closer to where data is produced. It is envisioned as a massively geo-distributed cloud infrastructure, where endpoints are placed there where are most needed to bring ultra-low round-trip latency to users. However, every new edge location encompasses a new set of unique features that shapes the size and type of resources to be installed. As such, edge cloud infrastructures are heterogeneous at their core and not by choice. In the edge, applications do not have the luxury of choosing on what hardware they prefer to run. In the edge, *what you see is what you get.*

1.1 Context & Motivation

In the early days of computing, batch processing gave way to time-sharing, allowing businesses to provide their employees with enough compute resources for their day-to-day duties. However, with the advent of personal computers, businesses lost the incentive to invest in vast and expensive mainframes. Everyone was already getting enough compute and storage to work on their documents and spreadsheets. Moreover, as the world entered the 21st century, the Web 2.0 [26] gave users an active role on the internet revolution. A role users still have. Quickly, the need for more and faster compute and storage resources began to shift from the users' devices to the service providers' infrastructure. Businesses would over-provision their infrastructure to cope with peak demands, yet these peaks could span anything from a few hours a day to even just a few days a year. The rest of the time, these infrastructures were mostly idle. Average utilization of 5% to 20% was normal at the time [73, 86]. On top of that, the increasing operational costs only made matters worse.

Amazon was the first major business to realize there was a huge problem to be solved if the internet was to become the enabler of the 21st century's economy. In 2002, Amazon launched the first iteration of their Amazon Web Services. In 2006, storage and elastic computing services would join to start with the pay-as-you-go model, which still dominates the market today. With it, a new era of public datacenters began.

1.1.1 Ascension to the Cloud(s)

Cloud computing emerged out of the necessity to keep scaling the infrastructure up while facilitating its management and maintenance. Thanks to the centralization of compute and storage, Cloud Service Providers (CSPs) have been able to provide elastic access to these resources over a wide area network (WAN). Moreover, as Internet Service Providers (ISPs) around the world deployed faster and cheaper broadband connections, internet services saw an unprecedented growth that made the internet the center of a new economy that revolves around data. Thanks to this, CSPs enjoyed high levels of user aggregation that allowed them to keep growing and reduce costs.

However, the landscape of devices connected to the internet and their access patterns is changing quickly. In 2005, a mere 16% of the global population (587 million) had access to internet [97], while in 2019 this number increased to 53.6% and is expected to grow to 75% by 2025 [75]. At the same time, the amount of data generated in real-time is expected to grow up to accounting for 30% of the total amount of data generated (doubled with respect to 2017).

After years of unlimited expansion, the cloud can be considered a decentralized mesh of centralized compute and storage clusters. Albeit extremely powerful and versatile, the cloud is starting to show its flaws. The internet is no longer what it used to be when the cloud was first envisioned, and the overhead of reaching resources hundreds of kilometers and several network hops away starts to become a burden for modern use cases that need to keep growing.

1.1.2 A new world demands a new Cloud

As of 2021, Amazon provides its services to Europe from just 6 [10] geographical locations and Microsoft Azure from 7 [11]. Consequently, today, the round-trip latency of a simple request that needs to reach a cloud location is dominated by the network latency to reach the compute node and not by the computation done at arrival [95]. Data centers are often hundreds of kilometers away with many network hops in between and double-digit latency is the norm, while a single-digit is harder to see than a three-digit latency. At the same time, the advent of 5G broadband infrastructures are democratizing sub-millisecond access to medium latencies. Such low latency enables new use cases that were unthinkable just a few years ago, mostly due to unfeasible latency requirements at the time. For example, Advanced Driver-Assistance Systems (ADAS) together with Smart Cities require information to reach cars traveling at high speeds; Virtual Reality (VR) or Augmented Reality (AR) require new images to reach the user in 30 milliseconds intervals (60 fps) or the user experience is ruined. Moreover, these use cases generate vast amounts of network bandwidth that are bounded to their geographical location. That is, autonomous cars, for example, compute their actions based on the information from nearby cars and cameras but are oblivious to whatever happens a few streets away. There is no need for this information to travel hundreds of kilometers to be computed in a centralized data center while unnecessarily consuming backhaul and core network bandwidth. Other use cases, such as those involving private healthcare information from patients, cannot be moved to the cloud due to security and privacy reasons. Altogether, the only foreseeable solution relies on moving compute and storage resources closer to where the data is produced, i.e., closer to the end-user. This is where the edge cloud comes into play. Because the edge cloud is *the only foreseeable solution* [84, 12, 61, 82, 37, 62, 3, 9].

1.1.3 Descend to the Edge

The edge cloud is a geographically distributed infrastructure that provides compute and storage to users in a similar fashion to current cloud infrastructures. Edge computing is a distributed computing paradigm in which compute and storage resources are moved to the edge of the network. The definition of *edge* varies depending on the agent deploying the infrastructure. Telecommunication Service Providers (TSPs) consider the edge to be a point close to the end-user but still under the TSP's infrastructure and control, while service providers refer to the edge as the different locations from which their services are served. Sometimes, certain IoT devices or other end-user devices such as smartphones with (limited) compute capabilities are considered to be the edge of the network, but these have different design challenges. In this thesis, we consider the edge to be devices that are part of the infrastructure of any TSPs or enterprise but not the end-user, as the latter involve a different set of challenges.

Shifting resources to the edge of the network promises to deliver low latency while relieving backhaul congestion. As a result, the edge cloud has been appointed as a key enabler and, in some cases, as the *only* realistic approach to meet real-time requirements of some use cases while allowing services to keep scaling cost-effectively [84, 6]. In its simplest form, what the edge cloud provides is similar to what Content Delivery Networks (CDNs) have provided for years. CDNs are, however, a small part of what the edge cloud aims to deliver; a single service among the various services that can benefit from compute and storage deployed at the edge of the network. Therefore, the edge cloud is conceived as a heterogeneous infrastructure by nature. Different services with different requirements and constraints are deployed at different geographical locations to cover different populations and areas, while different physical locations may have different constraints regarding the type and size of the hardware to deploy. For example, a street cabinet cannot fit as many nodes as a Data Center, while connectivity and cooling options are also more limited. Consequently, the level of user aggregation greatly differs from one location to another and narrows as we move further to the edge. This implies that deployments must be carefully planned according to the specifics of every location. With edge locations that spread over a wide area and the operational costs involved, under-provision, over-provision, or a simple mismatch between the hardware deployed and the hardware needed become expensive inefficiencies compared to those in centralized cloud locations, where user aggregation levels that are orders of magnitude larger allow for fine-grain adjustments through the software management layers. Simply put, in the cloud, whatever challenges workload heterogeneity poses are mitigated by installing enough resources for everyone and letting its users decide what set of resources they will need for their application. User aggregation will do the rest to achieve high resource utilization. However, in an edge where resources are scarce, edge cloud service providers must plan ahead and dimension deployments taking into considerations the kind of workloads that will run from such deployments.

1.1.4 New Cloud, New Challenges

A massively distributed and heterogeneous infrastructure brings as many opportunities as challenges [98, 2, 33]. On one hand, edge cloud infrastructures are to bring resources closer the where the data is produced. That is, part of the load that is currently borne by centralized locations in the cloud will shift towards multiple and geographically distributed points in the edge of the network. This promises to enable lower latency access to compute and storage resources while aleviating congestion in the backhaul. Altogether, the edge is expected to enable modern use cases while allowing existing ones to keep scaling at the current pace or even faster.

On the other hand, edge cloud infrastructures are to provide a frontline for compute and storage resources that will sit between the user and current cloud infrastructures. A frontline that will be unavoidably limited by the amount of resources and, thus, what it can process. Therefore, applications must also shift from a paradigm in which resources are virtually unlimited to a new paradigm in which the available resources can become the bottleneck.

Therefore, the cloud and the edge cloud must agree upon who should take care of what. For example, the edge cloud is expected to mostly assume use cases with a specific set of characteristics, such as real-time, low latency, or filter the information that makes it to the cloud. This will undoubtedly shape the workloads and their resource requirements in both sides.

Service and resource management layers that are aware of the locality and heterogeneity in the edge [51], privacy and security [31], or simply decide on the optimization metrics are just some of the challenges that arise along the edge cloud [84, 33].

A successful edge cloud deployment requires to first understand what are the specific set of requirements and restrictions that an heterogeneous infrastructure entails. Once deployed, the transition to the edge should be as seamless as possible through the interplay between the traditional cloud and the edge cloud. Finally, it is desirable for the edge to reduce its dependance from the cloud as much as possible through new methods and management layers that can leverage an heterogeneous and geographically distributed infrastructure.

Thus, we can summarize the existing challenges towards a successful large-scale adoption of the edge cloud as follows:

Characterizing Edge Heterogeneity. In the edge, heterogeneity is on the rise. Each location entails a different set of requirements and restrictions (dependant on the geographical location, the space available, the coverage area, and the services to provide) and this must be properly understood before we can begin to design the infrastructure. Only through a thorough analysis and characterization of the implicit heterogeneity present in the edge will we be able to efficiently define and shape efficient edge infrastructures.

Transitioning from Cloud to Edge. Compute is compute and storage is storage no matter where they are. However, these two must be treated differently as we transition from an unrestricted cloud towards a resource-constrained edge. In the edge, resources are scarce and should be treated accordingly. In return, edge environments provide a series of traits, such as a high locality of data, services and users, that can be leveraged through an application-infrastructure co-design and will allow us to adapt and optimize applications for such resource-constrained environments. Nonetheless, there is a limit to what such nodes may offer. Luckily, there where the edge does not reach, the cloud does. Therefore, if we expect a seamless transition to the edge, we first need mechanisms that can leverage the full potential of the edge and, then, provide an interplay between the cloud and the edge.

Distributing work among Edge locations. As a massively and geographically distributed infrastructure, the relatively low level of user aggregation in the edge cloud (as opposed to the cloud) poses a challenge to achieve high levels of resource utilization and, ultimately, to be a cost-effective solution. Therefore, new mechanisms are required to distribute and balance the work between edge locations and mitigate this issue.

This thesis addresses the aforementioned challenges to easen the transition from centralized cloud infrastructures to massively distributed and resource-constrained edge cloud infrastructures.

1.2 Contributions

The main goal of this thesis is to demonstrate that we can optimize edge cloud deployments for video analytics through new mechanisms that can leverage a heterogeneous, geo-distributed, and resource-constrained set of locations which are expected to provide compute and storage to a specific geographic region, namely the edge cloud. To this end, we:

C1 Present an in-depth characterization of end-to-end video analytics executed on a heterogeneous set of platforms.

C2 Develop and evaluate a novel automation framework to optimize neural networks for video analytics through Edge-Cloud interplay.

C3 Develop and evaluate a novel technique allowing to optimize and distribute video analytics workloads among edge nodes.

Figure 1.1 summarizes the contributions of this thesis placing their logical representation along two axes: scale and functionality. Each contribution builds on the previous one and evolves on the scale and functionality it provides. The first contribution starts from a sandbox environment to understand the landscape of the hardware platforms to be considered in different edge locations to provide video analytics from the edge are susceptible to conform the Edge Cloud along with their requirements and restrictions. From there, the second contribution focuses on how a single edge location can be assisted by a centralized cloud. Finally, the third contribution considers a fully distributed edge cloud that is able to break video analytics down into smaller work units to enable the distribution of work among locations.

1.2.1 C1. Characterization of Video Analytics in the Heterogeneous Edge

Every single edge cloud deployment is potentially unique. Together, the geographical region to which to serve, the physical space in which resources are installed, the services running from it, or the expected user aggregation, are some of the aspects that will define the optimal size and type of the resources to install.

Among all the services to serve from edge locations, real-time video analytics stands out from the rest due to its strict latency requirements and the amount of bandwidth it

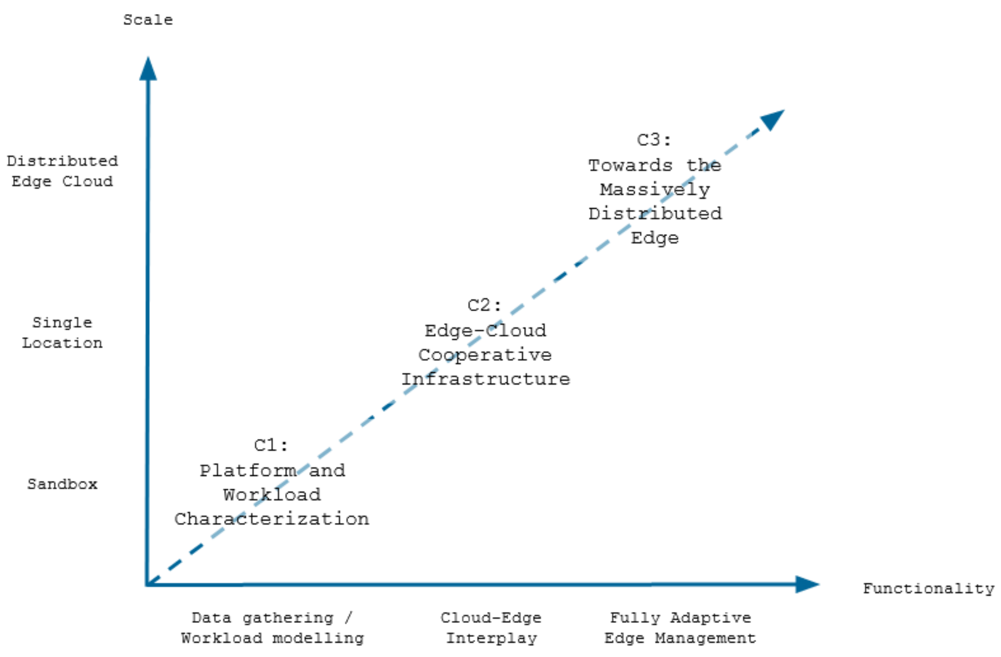

Scale
Distributed Edge Cloud
Single Location
Sandbox
C1: Platform and Workload Characterization
C2: Edge-Cloud Cooperative Infrastructure
C3: Towards the Massively Distributed Edge
Functionality
Data gathering / Workload modelling
Cloud-Edge Interplay
Fully Adaptive Edge Management

Figure 1.1: Summary of the contributions of this thesis.

generates [6]. However, video analytics is a compute intensive task whose typical pipeline entails two equally intensive tasks: media decoding and execution of neural networks. While both tasks are a potential bottleneck, the variables that impact their performance are different. Decoding performance is tight to the encoding and resolution of the input stream. If these two are known, the resource requirements can be approximated with high accuracy. However, inference performance depends on multiple variables that span over multiple dimensions: neural network architecture, number of layers and parameters, type of operations, etc. Even if two distinct networks are solving the same type of problem (e.g., car detection), a single computing device could execute thousands of inferences per second with one network and less than a single inference per second on the other, all depending on the internals of the network. Therefore, we need full understanding of the intricacies of video analytics if we expect to anticipate its requirements in different scenarios.

The first contribution of this thesis presents a thorough characterization of end-to-end video analytics executed on a heterogeneous set of hardware platforms (including media, and neural network accelerators). The goal of this contribution is to understand how each of different hardware platforms may become the preferred platform when considered under varying constrained scenarios. Consequently, latency, throughput, and power are all analyzed and used to estimate end-to-end metrics.

The work performed in this area resulted in the following publication:

Daniel Rivas, Francesc Guim, Jordà Polo, and David Carrera. **Performance Characterization of Video Analytics Workloads in Heterogeneous Edge Infrastructures**. *Concurrency and Computation: Practice and Experience*, page e6317, 2021 [80].

Moreover, the challenges addressed in this thesis required us to explore the use of different paths, including FPGAs that can be disaggregated and shared by multiple users in edge cloud environments. The work performed in this area resulted in two granted patents and one patent application (i.e., under review) that propose improvements on the multi-tenancy of FPGAs accelerators.

Francesc Guim Bernat, Daniel Rivas Barragan, Karthik Kumar, Thomas Willhalm, Patrick Lu. **System, Apparatus And Method For Multi-kernel Performance Monitoring In A Field Programmable Gate Array**, March 26, 2019. *US Patent 10,241,885.*

Francesc Guim Bernat, Daniel Rivas Barragan, Karthik Kumar, Thomas Willhalm, Patrick Lu. **Device, System And Method To Access A Shared Memory With Field-programmable Gate Array Circuitry Without First Storing Data To Computer Node**, April 7, 2020. *US Patent 10,613,999*

Francesc Guim Bernat, Daniel Rivas Barragan, Suraj Prabhakaran, Kshitij A. Doshi. **Technologies For Opportunistic Acceleration Overprovisioning For Disaggregated Architectures**, November 10, 2020. *US Patent App. 20,190,102*

These patents highlight how research is not a straight path, and exploration on one side can lead to identifying a previously unnoticed problem or solution and result in a new contribution on a different research line. Moreover, these patents serve as motivation and justification to reconsider FPGAs for video analytics in a multi-tenant edge cloud.

1.2.2 C2. Edge-Cloud Interplay for Rapid Large-Scale Edge Deployments

The edge is not envisioned to replace the existing centralized cloud but to assist it to provide lower latency to the end-user and reduce saturation of the backhaul. Therefore, the edge and the cloud must work together towards this goal and not independently. The cloud can provide seemingly unlimited resources while the edge provides a layer of compute to process data with high locality that can be made aware of its context.

The work in this contribution is based on the premise that video analytics workloads should exploit context-awareness granted by edge locations if they are to execute high-accuracy neural networks on locations with limited compute resources. For example, object detection models (commonly used in video analytics) are usually evaluated based on their accuracy. However, accuracy often comes at the cost of a higher computational complexity. Figure 1.2 exemplifies the clear trade-off between speed and accuracy of different neural network architectures for object detection. As a result, real-time performance on resource-constrained nodes is limited unless they give up on accuracy or vice versa. But these numbers have a catch. On the one hand, challenges designed to evaluate state-of-the-art accuracy usually focus on a neural network's ability to generalize to a myriad of different object classes. For example, the COCO dataset (on which the models in Figure 1.2 were evaluated) contains objects so dispair such as airplanes, giraffes, and toothbrushes. On the other hand, we can safely assume these objects will rarely all appear together in the same scene [1]. These challenges offer an excellent method to standardize the evaluation of neural networks and allow fair comparisons among them. However, the different neural network architectures should converge towards maximum accuracy as the task gets simpler. Through awareness of the context of a camera for video analytics deployed to the edge, we aim to reduce the search space to either reduce the computational requirements or boost accuracy.

The second contribution of this thesis presents and evaluates **COVA** (*Contextually Optimized Video Analytics*), a framework that provides a mechanism to fully automate the

[1]The COCO dataset does not, indeed, contain any example of an airplane, giraffe, and toothbrush together in the same scene [24]

entire end-to-end pipeline from analyzing contextual information from cameras to specializing models for video analytics. Once a camera is deployed, COVA captures images and selects the most representative. Then, these images are annotated by a *groundtruth model* being executed in a remote data center. The groundtruth model acts as an oracle and its predictions are taken as the groundtruths during the training of the edge model. By assuming that images are only from static cameras, the framework can make use of traditional computer vision techniques to determine the representativeness of each frame. At the same time, these same techniques are also used to boost the accuracy of the groundtruth model and improve the quality of the resulting edge model.

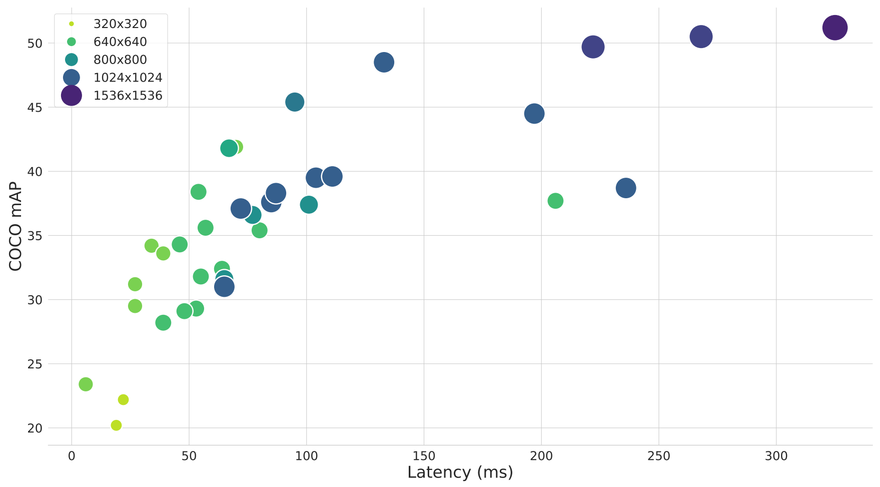


Figure 1.2: Latency (ms) versus mAP (mean Average Precision) on the COCO dataset of pre-trained object detection models available as part of the TensorFlow Model Zoo [94].

The work performed in this area has resulted in the following manuscript:

Daniel Rivas, Francesc Guim, Jordà Polo, Josep Ll. Berral, Pubudu M. Silva, and David Carrera. **Towards Automatic Model Specialization for Edge Video Analytics**. 2021. *Manuscript submitted for publication to Future Generation Computer Systems journal and is currently under review* [79].

1.2.3 C3. Towards the Massively Distributed Edge

The edge cloud comprises a growing number of compute nodes that are often tight on resources, especially the closer they are to the edge of the network. As such, applications running in edge nodes should leverage every bit of compute regardless of its location within the infrastructure. For this, applications must first be able to break their work down and only move the bare minimum around or, otherwise, the network is at higher risk of saturation as the deployment grows.

The third contribution of this thesis presents a novel technique to improve the efficiency of neural networks by filtering most of the scene out. Moreover, this technique allows the workload to be distributed among other nearby nodes. The work on this contribution is based on the following simple observation within the context of static cameras: *scenes do not move, objects (of interest) do*. Through this observation, we can assume that only those regions of an image that have changed contain objects that have to be identified. Consequently, we are able to narrow the search space, focus on such regions, and discard the rest of the scene.

The work performed in this area resulted in the following manuscript:

Daniel Rivas, Francesc Guim, Jordà Polo, Josep Ll. Berral, and David Carrera. **Large-Scale Video Analytics through Object-Level Consolidation**. 2nd EAI International Conference on Intelligent Edge Processing in the IoT Era, 2021[2] [78].

1.3 Thesis Organization

The remaining chapters of this document are organized as follows. Chapter 2 presents core concepts that are the backbone of this thesis. Chapter 3 presents a detailed characterization of video analytics along with an heterogeneous set of hardware platforms. Chapter 4 presents and evaluates a framework aimed at speeding up edge deployments through the end-to-end automation of the task to specialize neural networks for video analytics to the context in which an edge camera will be deployed. Chapter 5 presents and evaluates a novel method to enable the distribution of video analytics workloads while increasing efficiency and accuracy of the models without requiring any training process. Finally, Chapter 6 presents the conclusions and the future work of this thesis.

[2]This article received the *Best Paper Award* of the conference [1].

Chapter 2

Background

The purpose of this chapter is to provide the theoretical background on which this thesis is grounded. First, we take a closer look at the edge cloud, how its infrastructure is being envisioned (and deployed), and how this will shape the way services can be optimized for it. Following, we introduce neural networks and their relationship with the edge. Neural networks come in any form and size, depending on the type and scope of the problem they solve. We might not know what neural networks will accelerate the edge just yet, but one thing is certain: neural networks will be at the heart of the edge cloud. They are expected to drive, at least in part, most of the use cases of the future, and the edge cloud must be prepared to execute them efficiently and at scale regardless of their type and size. Finally, we delve into video analytics to understand why it is gaining interest from industry and academia and why it needs the edge cloud to be successfully deployed at large-scale.

2.1 The Edge Cloud

The edge cloud (or simply the *edge*) is a geographically distributed architecture that brings compute and storage resources to the edge of the network, i.e., closer to the end-user and closer to where the data is produced. With it, edge computing arises as a new form of distributed computing with the promise to improve response times and save network bandwidth compared to cloud computing, which relies on centralized compute locations far from the user.

The definition of *edge* varies depending on the agent deploying the infrastructure. For example, telecommunication service providers (TSPs) consider the edge to be a point close to the end-user but still under the TSPs' infrastructure and control, while service providers refer to the edge as the different locations from which their services are served. Sometimes, certain IoT devices or other end-user devices with limited compute capabilities are considered to be the edge of the network, but these have different design challenges. In this thesis, we only consider devices that are part of the infrastructure of any TSP or enterprise, but not the end user.

The edge cloud, as an architecture bringing resources one hop away from the user, was recognized as a way to fully integrate compute-intensive capabilities into our society as early as 2009 [81]. Back then, the term *cloudlet* was coined to refer to instances of cloud-like infrastructures deployed at the edge of the network. During the last years, the edge cloud has gained traction and has been appointed as the key enabler for multiple future use cases. The most commonly referenced are those use cases that are highly sensitive to latency, such as autonomous driving [61], or virtual reality [12]; those that continuously generate data, most of which is to be consumed in real-time, such as IoT devices monitoring in manufacturing [37];

or those that cannot rely on a shared centralized cloud due to working with highly sensible data, such as personalized healthcare [9]. However, among all, video analytics stands out from the rest for multiple reasons, e.g., strict latency constraints, large amounts of data to be consumed in real time, and can be considered a basic building block used to create a myriad other use cases (more in Section 2.2).

Nonetheless, the edge cloud presents new challenges that will have to be addressed on new edge deployments [84, 31, 2] while each of the different use cases to accelerate from the edge has its own set of challenges [54, 61, 12, 37, 9]. Specifically, in this thesis, we focus on how the many possible geolocations make edge deployments subject to a series of limitations or constraints that are not shared with traditional centralized data centers and are virtually unique for each location. This results in the different locations creating heterogeneity at many levels: heterogeneous hardware to provide service to a heterogeneous set of applications from a heterogeneous set of locations. Together, heterogeneity will set and push the constraints and requirements to be considered before any new edge deployment. For example, street cabinets, remote locations powered by solar panels, or rural areas with only satellite connectivity are all constrained by different factors (i.e., physical space, power budget, or thermals) but, at the same time, will also have a different level of user aggregation. Therefore, the orchestration layer must relate each location to its compute capacity on a specific workload to optimally allocate resources while meeting latency constraints [89]. Consequently, this thesis assumes a hierarchical edge architecture divided into three tiers; each tier increases capacity and aggregates services from the tiers beneath. Such multi-tier architecture has proven to increase overall system's efficiency while reducing the average response time [96].

2.1.1 Edge Architectures

Similarly to traditional cloud infrastructures, edge cloud infrastructures are designed based on multiple factors that span from expected number of users to budget. However, the consensus seems to be heading towards a hierarchical tree-like architecture whose leaves are nodes that are more constrained but also the closest to the end-user [96].

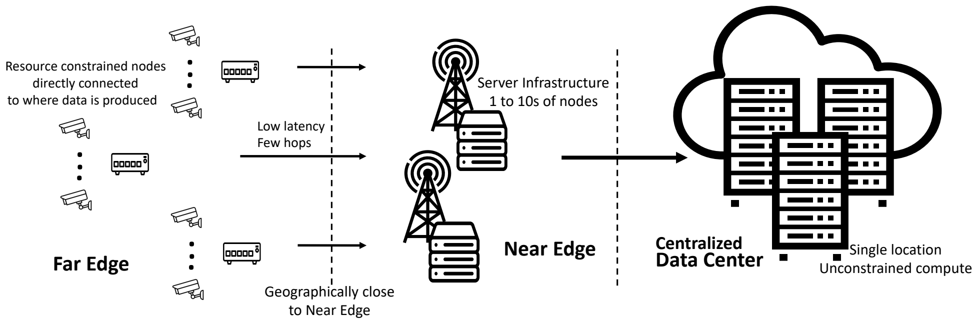

Resource constrained nodes directly connected to where data is produced
Low latency Few hops
Server Infrastructure 1 to 10s of nodes
Far Edge
Geographically close to Near Edge
Near Edge
Centralized Data Center
Single location Unconstrained compute

Figure 2.1: Example of a 3-Tier edge architecture.

Throughout this thesis, we consider a hierarchical edge architecture [96] with three *logical* tiers of compute locations[1] whose characteristics are primarily determined by their proximity to the end-user: Data Center, Near Edge, and Far Edge. We also consider that their proximity to the end-user will determine, or at the very least influence, the physical locations where said infrastructures can be deployed. The physical location, however, determines a series

[1]The infrastructure may have more than three levels of compute but the logical definition divides them all in only three tiers.

of aspects that narrow the number of possible hardware configurations down. The three categories are defined as follows:

Data Center Centralized. Regularly, data centers are only limited by the budget and are usually sized to aggregate the maximum number of clients. These deployments may include any kind of accelerators (GPU, FPGAs, etc.).

Near Edge Deployed in street cabinets. Small form factor to 1U server nodes. Power might be limited to what a regular outlet can provide (1800W to 2400W), while size is limited to what a street cabinet can fit, around 1 to 5 nodes. Air-cooled. 10 gigabit Ethernet is common for these type of nodes. These deployments may include up to one low-profile PCIe accelerator.

Far Edge From IoT gates to a single desktop processor in a small form factor. Total power is limited to 50W-100W due to cooling and thermal limitations. These nodes are designed to be in the open and subject to solar radiation while working with no more than a small fan or even passive cooling in some cases. These nodes are often used as WiFi or Broadband access points. Therefore, their connectivity is limited, ranging from a few hundred megabytes wireless up to gigabit Ethernet. The range of accelerators that can be installed in these nodes is also limited to small low-power USB or M.2 accelerators. On the other hand, media acceleration on these platforms is common.

More precisely, these categories are defined by their proximity to the end-user along with the difficulties associated with such proximity. In other words, a room full of servers in the middle of a metropolis does not fit under the Far Edge definition only because it is close to the users nearby, as it was probably designed with unrestricted access to resources in mind. Table 2.1 summarizes some of the technical restriction each category entails. Moreover, Figure 2.1 depicts an example of a 3-Tier edge architecture with the composition of nodes in each tier. From left to right, the Far Edge is composed by multiple and geographically distributed small nodes that are directly connected to the source producing the data. Near Edge locations aggregate multiple Far Edge nodes and are only a few hops from them. In urban areas, Near Edge nodes can be deployed at the edge of the backhaul network, which gives enough aggregation, space, and isolation to place a server infrastructure with multiple nodes. Finally, a traditional centralized Data Center with no theoretical upper limit on size or power but consequently limited to handful of locations per country or even continent, with the corresponding cost in latency.

	Form Factor	**Max. Nodes**	**Power Budget**	**Accelerator**	**Cooling**
Data Center	Blade Server	10-100	Any	Any	Any
Near Edge	SFF / Blade	5-10	<1KW	Low Profile PCIe	Air
Far Edge	IoT Gate	1-5	<50W	SoC / USB / M.2	Passive

Table 2.1: Technical specifications and restrictions for each infrastructure category.

2.2 Video Analytics

Video Analytics (VA), also known as Video Content Analysis (VCA), is the process of automatically detecting and recognizing spatial and temporal events in videos. Thanks to the recent advances in computer vision (boosted by the parallel progress in the field of machine learning), video analytics already drives most of the efforts towards automating tasks previously reserved to humans. On-board cameras help autonomous cars, as well as most UAV's (Unmanned Aerial Vehicle) understand their surroundings and take appropriable action; a simple laptop's camera can derive in new human-machine interaction through gesture recognition; and static cameras can analyze a city's traffic in real-time to do anything from finding a parking spot to warn emergency services upon accident detection, all while proposing drivers take an alternative route.

The edge cloud has been deemed as the key enabler for video analytics [84] mainly due to two reasons: low-latency requirements and high-bandwidth consumption. In some use cases, low latency is desirable to maintain certain QoS level. In others, guaranteed low latency is critical to avoid catastrophic failure while triggering a safeguard. Nonetheless, they all have one common denominator: the amount of data being constantly generated. In an hour, a single low-end camera generates above 1.5 GB[2]. Moreover, most use cases require data to be consumed real-time. Therefore, the use of centralized and distant data centers becomes far from optimal. Consequently, the edge cloud has been appointed as the *only* realistic approach for real-time video analytics [6].

However, video analytics arises one more problem to resource-constrained Edge locations. In addition to latency and bandwidth, video analytics has considerable compute requirements. The typical pipeline of video analytics applications entails two tasks that are comparably compute intensive, namely video decoding and execution of neural networks. Figure 2.2 depicts an example of a common video analytics pipeline. A video feed is received as input. Video frames are decoded at a given frame rate and the outcome is an RGB matrix containing the color information of each pixel. This matrix is then passed as input to the first neural network. In the example, *NN1* performs vehicle detection. Its results, in the form of bounding box coordinates for all vehicles detected in the scene, are used to crop regions from the original frame. These regions are passed to two other neural networks. On one side, *NN2.1* detects license plates and *NN2.2* applies character recognition to the license plates detected. On the other hand, *NN3* extracts visual attributes from the images of the detected vehicles.

2.2.1 Video Analytics Systems

Video analytics applications can be divided into *online* and *offline* video analytics, depending on *when* information extracted from the frames is needed. On the one hand, online video analytics requires images to be processed in real-time, as actions are triggered with respect to what is in front of the camera at the moment images are captured. Automatic license plate recognition to control a barrier to a given facility or an autonomous car detecting obstacles to avoid them would fall under this category. At the same time, past information is no longer useful. Therefore, such systems are typically evaluated in terms of inference latency but are unable to fully exploit parallelism due to the *hazards* of request consolidation with strict deadlines. On the other hand, offline video analytics processes past images, usually to allow

[2]Numbers for an HDTV camera (1280x720 pixels) at 25 frames per second. At FullHD (1920x1080 pixels) generates between around 3.5 GB per hour and around 15 GB if images are captured at 4k (3840x2160 pixels) [104]

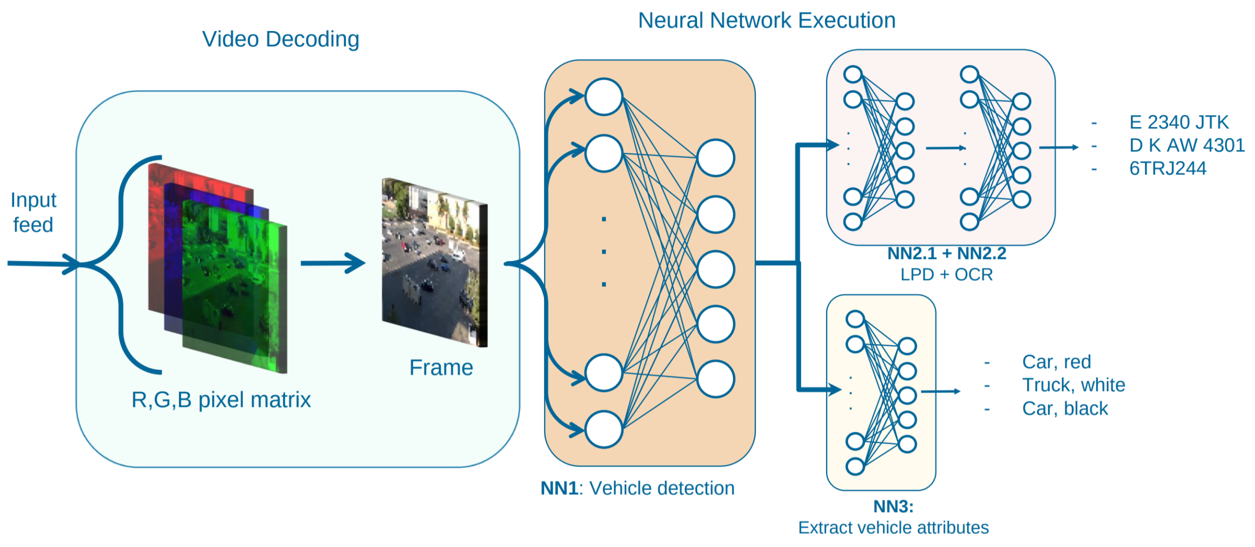


Figure 2.2: Example of video analytics pipeline. After a video frame from a scene is decoded and preprocessed, it is passed to one or more neural networks to extract information from the contents of the frame. Detections from one model may trigger further analysis on subregions of the scene.

querying video and obtain the fragments that contain the queried objects. These systems analyze images in bulk and can, therefore, fully utilize resources, as the primary evaluation metric is the latency of the queries (i.e., total system throughput). These two categories have very distinct requirements.

This thesis focuses on online video analytics for three reasons. First, they generally show a subpar scalability in terms of performance and efficiency as having to meet strict latency deadlines limits the extent to which some optimization can be applied. Second, offline video analytics can also benefit from an online analysis to pre-process images and reduce the search space of future queries [43]. Third, while optimizing queries is important, off-line video analytics tolerates higher round-trip latencies, which, if needed, allows for the workload to be off-loaded to a centralized data center with access to hardware acceleration.

2.2.2 Real-Time Video Analytics

Real-time video analytics differs from other types of video analytics in that the results of the analysis are used for real-time decisions. This has two implications on how users and services have to be considered from the system orchestrator's point of view. First, videos generated from cameras must be constantly analyzed, even if most of the contents are irrelevant. Second, latency and throughput are usually strict constraints attached to every use case, such as traffic control, security, or digital assistants.

However, other types of video analytics that continuously analyze images have been proposed as means of pre-indexing the contents of frames to speed-up future potential queries over certain footage [43]. Although very similar, this type of analysis is more tolerant to latency and therefore give orchestrators more freedom to dynamically allocate resources depending on whatever is available at a certain moment.

2.3 Neural Networks

2.3.1 Neural Networks for Video Analytics

While not inherently coupled with the use of neural networks, it is thanks to them that video analytics is becoming increasingly popular and, already today, is used in all kinds of production systems to provide an automated analysis of a sequence of images. Such analyses can yield different types of useful information (e.g., classification, detection or recognition). This thesis, however, focuses primarily on the task of object detection (and image classification to a lesser extent), for which Convolutional Neural Networks (CNNs) are the *de facto* standard. Nonetheless, its contributions and conclusions should be extensible to other types of analysis as long as they also use CNNs.

Object detection is the task of locating and identifying objects within an image or scene and is, in fact, a combination of region proposal and image classification problems. Region proposal techniques suggest a set of regions in the input image that might contain something of interest (i.e. something the neural network was trained to detect). These regions are then processed as independent images and are classified based on their contents. The region is considered a positive example if the highest scoring classification for that region is above a user-defined threshold. Intuitively, region proposal mechanisms have a high impact on the quality of the detection results. There are several methods available, such as Selective Search or Focal Pyramid Networks (FPN)[59]. Neural networks that have differentiate region proposal and classification in two stages are called *Two-Shot* detection models. On the contrary, *Single-Shot* detection models skip the region proposal stage and yield localization and class predictions all at once. Regardless of the number of shots, all object detection models predict bounding boxes and classes using a single input layer of a fixed size decided during training. Larger input layers potentially yield more accurate predictions, as more pixels (i.e. information) are taken into account, especially for smaller objects that are more difficult to correctly detect. However, any increment in the input layer's size is followed by an increment in compute requirements that is not always matched with an equal increment in accuracy.

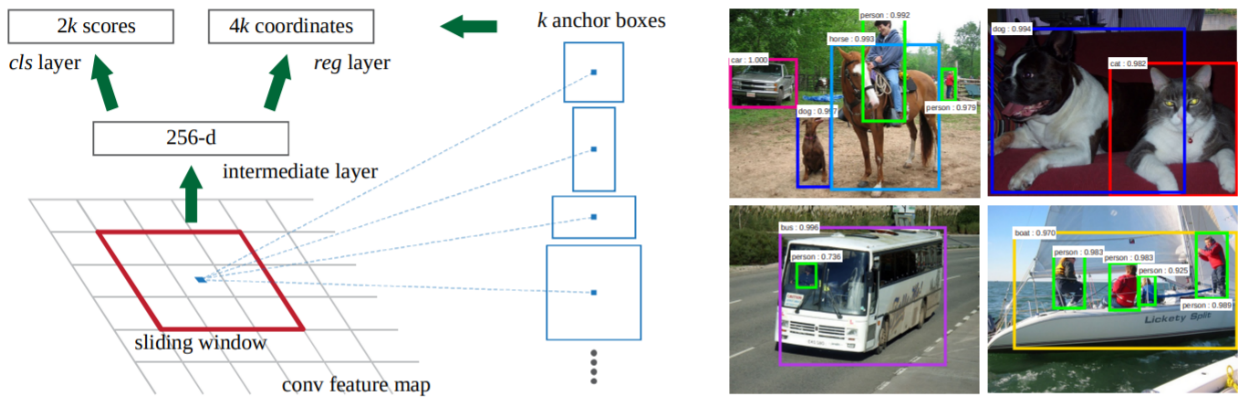


Figure 2.3: Left: Region Proposal Network. Right: detected objects from the VOC2007 dataset. Source: [76].

2.3.2 What affects the accuracy of a model?

The resulting accuracy of a neural network model after training is affected by numerous factors, the most obvious the quality of the dataset and the chosen hyperparameters. The training dataset must be large enough and present enough variety to reasonably approximate the unknown underlying mapping function from inputs to outputs, while the hyperparameters have direct control over the learning process. We can consider hyperparameters to be anything that impacts how the model learns and whose values can be manually set. They differ from regular parameters in that they are external to the model, i.e., their values are set before the model begins to train and cannot be changed during the learning process. The neural network architecture to train (number, type, and size of the layers) is, probably, the first hyperparameter to decide but the learning rate of the gradient descent, the train-test split ratio of the dataset, the cost function to use, or the batch size (number of training examples used in each iteration) are also fundamental hyperparameters.

Nonetheless, other not-so-obvious factors will also affect the resulting accuracy of a model. For example, the receptive field and the size of the objects to identify are especially important in the edge, as edge cameras are prone to capture objects from afar and, thus, small.

Receptive Field

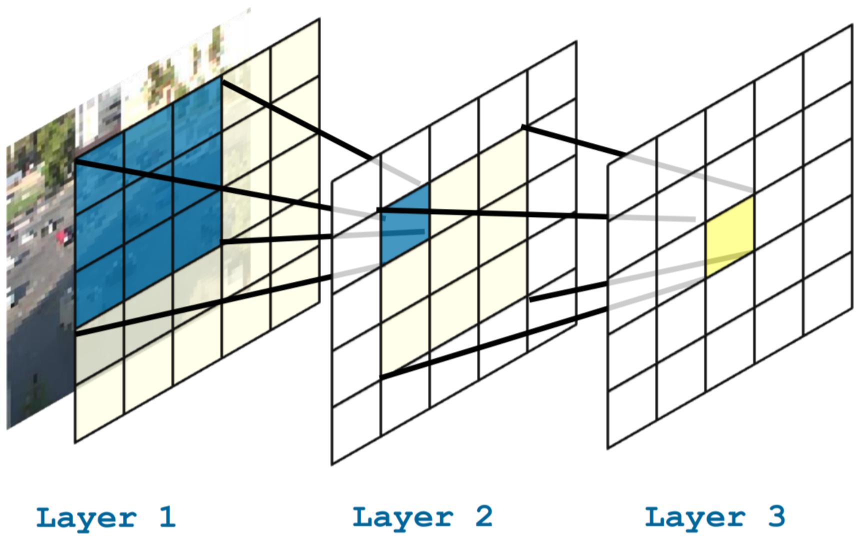


Figure 2.4: The receptive field of each convolution layer with a 3x3 kernel. The blue area marks the receptive field of one pixel in Layer 2, and the yellow area marks the receptive field of one pixel in Layer 3.

The receptive field is defined as the size of the region that produces a given feature, and it applies only to local operations, such as convolutions or poolings. Figure 2.4 depicts an example of the receptive field in two consecutive layers after applying a 3x3 kernel.

The idea of the receptive field is essential to understand how neural networks *see* objects, how the input size of a neural network affects the size of the objects that can be detected, and, ultimately, its accuracy.

Object Size Matters

The size of an object directly impacts a neural network's ability to detect it within a scene.

Neural networks for object detection see detection as a multi-task learning problem that combines classification and bounding box regression. Then, an intersection over union (IoU) threshold is required to define what constitutes positive and negative examples. The IoU will be 1.0 when the two bounding boxes (proposed and ground-truth) perfectly match and will tend towards 0 the more displaced the proposed bounding box is. Therefore, the smaller an object is, the more precise the bounding box regression must be. Conversely, a proposed bounding box for a small object that is a few pixels off has a high probability of yielding an IoU below the threshold and, thus, a false negative. Here is where the idea of the receptive field comes into play because the model cannot be more precise if it cannot see beyond a certain distance.

Moreover, there are two approaches to detection: Two-stage and Single-Shot detectors. Two-stage detectors, such as R-CNN, solve classification and bounding box regression in two different stages. This allows the bounding box regression to be more precise, although at a high computational cost. On the other hand, single-Shot detectors have a set of pre-defined anchors where the detector will try to find true positives. Therefore, Single-Shot detectors are even more susceptible to missing the smaller objects in the scene.

As authors in [21] state *"In general, a detector can only have high quality if presented with high quality proposals"*. In Chapter 4 and Chapter 5, we take this idea to boost the accuracy of the models but look at it from a different perspective to do it without increasing the amount of computation thanks to presenting the models with higher quality and higher definition regions of the scene using what we know from the scene.

Chapter 3

Video Analytics in the Heterogeneous Edge Cloud

In this chapter, we present the first contribution of the thesis, whose goal is to understand how the execution of video analytics is subject to the intrinsic heterogeneity that is present in the edge as a whole. That is, understanding the environment that edge cloud locations provide and the context in which video analytics applications will run, including all kinds of limiting factors down the line. This contribution aims to assist in the design of optimal heterogeneous infrastructures given the specifics of a deployment. These specifics are the physical location of the deployment, the set of workloads to run, and the expected user aggregation and load. Together, they define the best hardware configuration for an edge deployment.

Computer vision, thanks in part to the rapid advancements in Deep Learning, has boosted the use of cameras to automate the analysis and understanding of our surroundings. Video analytics is becoming increasingly popular, but its widespread adoption presents two major challenges: 1) the amount of data it generates, and 2) its computational complexity.

First, the amount of data that video analytic generates. An IP camera streaming a 1080p video generates around 0.5MiB in one second; the same camera will generate 1.4TiB of data over the span of one month. With the current state of the internet, where 70% of the network traffic corresponds to video [23], adding millions of cameras upstreaming content to data centers threatens to saturate backhaul networks. Therefore, the only foreseeable solution is for data to be processed close to where it is generated by means of deploying compute resources at the edge of the network. In response to this problem, the edge cloud has already been appointed as one of the main accelerators for video analytics [84]. The edge cloud promises to alleviate pressure (and costs) of the infrastructure while, at the same time, offer improved latency and overall QoS [17]. The edge cloud is defined as a decentralized and heterogeneous cloud where compute and storage is placed towards the edge of the network. Moreover, the further we move resources towards the edge of the network, the area of coverage becomes smaller and the requirements more specific. Therefore, the mix of hardware platforms to deploy can be tailored for each specific location.

Second, video analytics is computationally expensive. A common video analytics pipeline is composed of two main phases: video decoding and Neural Network (NN) inference. Traditionally, hardware platforms have been evaluated based exclusively on their performance for NN inference [55, 16], while overlooking the impact of feeding the networks. It used to be a reasonable approach as only a few years ago, state-of-the-art accelerators were not able to execute state-of-the-art NNs at more than a few frames per second [74]. The complexity and size of NNs have followed an upward tendency. However, NNs are not something as new or esoteric as they used to be and the tendency of improving accuracy by adding more layers is long gone. On the contrary, we now have a plethora of different NN architectures and optimization techniques at our disposal that widely expand the throughput-accuracy trade-off. For example, model distillation [42] proposes to use large and accurate NN, such as ResNet [40] or VGG [87], to teach much smaller NN models how to generalize. Moreover, model specialization trains small models that are only suited for specific tasks that can be assisted by large models in cases of uncertainty [55, 43]. Such techniques reduce, or even fully remove, the need to have complex generalist models in many scenarios, particularly in resource constrained environments, such as the edge cloud.

As NNs take over data centers, faster and more efficient hardware has been developed to accelerate NN execution. NVIDIA included Tensor Cores [64] in its GPUs, and even launched an inference-optimized GPU; Intel extended its AVX-512 units with Vector Neural Network Instructions (VNNI) to accelerate common operations on convolutional neural network algorithm [50]; FPGAs are trying to gain adoption as general-purpose accelerators thanks to their claimed high power efficiency when executing neural networks [47]; and other low-power accelerators have been released, such as the Intel MyriadX [48] or the Google Coral Tensor Processor Unit (TPU). The tendency to speed up NN execution is clear.

Overall, the improvements in hardware plus newer and faster neural networks are shifting the bottleneck from the inference to the rest of the pipeline. This is specially true for video analytics, since these improvements have not been correlated with improvements in video decoding, a task that is also computationally expensive [100]. This problem has been already highlighted in the literature [56], putting more resources into optimizing frame decoding to reduce overall time to solution of video queries. However, video queries assume bulk processing of videos, rather than frame by frame, and these methods rely on an initial exploration of the solution space to select the set of optimizations to apply depending on the time requirements of each query, which can take up to hours. Therefore, such methods have a different set of requirements and constraints than real-time video analytics in the edge, where latency deadlines are in the order of a few hundred milliseconds and the low level of user aggregation does not allow to consider any batch processing optimizations.

Processing video analytics on data centers in real-time can become cost prohibitive due to the amount of data it constantly generates and what that implies for the whole backhaul infrastructure. Therefore, the edge cloud must assist data centers if not completely overtake the task of processing. The heterogeneity of locations and requirements in edge deployments leads to one significant challenge: unlike traditional data centers, edge deployments are susceptible to be constrained due to non-technical factors (other than budget) that are often linked to the geographical location itself. In some cases, edge nodes will be constrained due to limited connectivity (e.g. only broadband or satellite connections available), limited power budget (e.g. solar-powered nodes), or even limited physical space (e.g. a small fog node installed on a streetlight). In other cases, edge nodes are installed for one reason (i.e., IoT gate) but can be re-purposed to accommodate video analytics (e.g. an IoT gate with a low-power USB accelerator). The *edge of the network* is a broad term and, in such heterogeneous landscape, we have to fully understand how each platform can best serve in each potential scenario. Consequently, the evaluation of edge platforms should not only consider quantitative metrics, such as performance or cost, but should also be considered within the context in which these platforms will potentially be deployed.

To address this gap, this chapter makes the following contributions. First, we propose a classification of hardware platforms based on their compute capabilities and the locations they are more likely to be deployed considering also power and size requirements. This classification is complemented with a classification of neural networks based on their size and expected use cases to solve. Second, we characterize six hardware platforms (including media and neural network accelerators), each falling into a different edge use case. Moreover, we also explore and evaluate FPGAs as a general-purpose accelerator for the edge. Finally, we extract the main conclusions of the characterization and introduce them into an experimental model. We use the resulting model to estimate performance and cost while demonstrating two things: considering video decoding and inference together have a considerable impact on the performance projections, and how that inevitably impacts the design of the optimal infrastructure for a given deployment. Hence, the design of edge infrastructures requires more accurate performance models that consider heterogeneity of platforms to its full extend.

This chapter is organized as follows. Section 3.1 revises the related work on the characterization of video analytics and, more specifically, video analytics in the context of edge cloud deployments. Following, Section 3.2.1, we describe the categorization of the edge cloud that we use to select the evaluation platforms. Similarly, in Section 3.2.2, we describe the categorization of the neural network architectures that we consider in our characterization. In Section 3.3, we describe the methodology followed throughout the chapter to characterize video analytics on the different hardware platforms. In Section 3.4, we present and analyze the characterization of video analytics, whose conclusions are summarized and used to estimate the end-to-end performance and cost of different potential use cases. Finally, Section 3.5 presents the final considerations of the chapter.

3.1 Related Work

Video analytics systems, while not specific to the edge, are key to achieve an efficient and cost-effective widespread adoption. In this chapter, we base most of our assumptions and considerations on top of what some of these systems have previously proposed [43, 55, 7], as described in Section 2.2.1. However, these systems focus on speeding up video querying and not on real-time video analytics.

With respect to the characterization and evaluation of video analytics, most of the work is focused on the evaluation of neural networks, while other phases of video analytics have been mostly overlooked. Authors in [44] provide an extensive analysis of the speed/accuracy trade-offs for several modern convolutional object detectors, whose goal is to serve as a guide for selecting a detection architecture depending on whether accuracy or speed is favoured over the other. A similar but more experimental analysis is presented in [16], where authors provide an extensive benchmark of the most representative deep neural network architectures for image classification and highlight the relationship between complexity and the size of a model with its speed. Execution of neural network, including CNNs, has been extensively characterized on both ends of the network. On datacenters, Researchers at Google [52] characterized three different datacenter platforms (Google's TPU, an NVIDIA K80 and a Haswell server-CPU) executing a mix of representative neural networks and showing how a specialized architecture can deliver better performance at lower cost. On the other end, resource usage on edge devices, including low-power accelerators such as Intel Myriad X, Google Coral, has been characterized by authors in [8]. However, to the best of our knowledge, none of these works have aimed to provide a characterization of end-to-end video analytics workloads for heterogeneous platforms.

With respect to previous evaluations, we add two ideas: (1) we evaluate a range of CNNs, not only those with state-of-the-art accuracy and (2) we analyze how these CNNs perform under different hardware, ranging from traditional data center platforms with power-hungry graphic accelerators all the way down to fanless CPU nodes with USB low-power NN acceleration. By combining these two to estimate the end-to-end performance, we can

	Processor	**Cores**	**Freq. (GHz)**	**Memory Peak BW**	**TDP**
Data Center	Intel Xeon Gold 6248[1]	20	2.5 - 3.9	384 GB (128 GB/s)	150 W
Data Center (GPU)	Intel Xeon Silver 4114[2]	10	2.2 - 3.0	64 GB (96 GB/s)	85 W
Near Edge	Intel Xeon D-2183IT	16	2.2 - 3.0	128 GB (96 GB/s)	100 W
Far Edge	Intel i7-8700T	6	2.4 - 4.0	16 GB (25 GB/s)	35 W
Fog IoT	Intel i5-8265U	2	1.6	8 GB (25 GB/s)	10 W

[1] Dual-Socket platform: Core count, bandwidth, and TDP are *per socket.*
[2] Dual-Socket platform: Core count, bandwidth, and TDP are *per socket.* TDP does not include GPU.

Table 3.1: Hardware platforms selected.

Device	**Acceleration**	**TDP**	**Interface**
Intel HD 620	Media	15 W	SoC
NVIDIA GeForce RTX2080	Media DNN	215 W	PCIe
Intel Myriad X	DNN	4 W	USB

Table 3.2: Media and DNN accelerators evaluated.

relate each platform to the type of model for which it is best suited.

3.2 Heterogeneous Edge

3.2.1 Heterogeneous Platforms

Following the taxonomy of Edge architectures presented in Section 2.1.1, we have selected a series of compute platforms to cover, at least, the three Edge location tiers. Consequently, Table 3.1 details the particular set of hardware platforms used and characterized throughout this chapter and their specifications. Moreover, video decoding and the execution of neural networks greatly benefit from the use of accelerators. As such, a thorough evaluation and characterization of Edge platforms should consider them. Table 3.2 lists the accelerators used throughout this chapter.

Category	**Model**	**Complexity (GFLOPs)**	**Size (MParams)**	**Input**	**Top-1**	**Top-5**
Reference	Faster R-CNN	57.203	29.162	600x1024		
	ResNet152	22.709	66.746	224x224	78.51%	94.45%
Edge	MobileNetv2	0.615	3.489	224x224	71.22%	90.18%
	SqueezeNet1.1	0.785	1.236	224x224	58.3%	81%
Specialized	ResNet16	0.702		112x112	-	-
	ResNet14	0.124		56x56	-	-

Table 3.3: Models evaluated on each category.

3.2.2 Heterogeneous Models

For the analysis, we select different state-of-the-art image classification CNN architectures that range in terms of complexity, and size. However, due to the huge amount of neural network architectures along with all the variables that can impact their execution (precision, pruning of layers, etc.), an exhaustive evaluation that considers every single architecture becomes unfeasible. Therefore, we have considered a taxonomy that divides neural network models into three categories according to the type of deployment they are expected to fill (based on size, speed, and accuracy) and evaluated two from each category:

Reference models Big and complex models that achieve state-of-the-art accuracy. These models are good at generalizing.

Edge models Smaller models with lower accuracy initially oriented to run at real-time speed (i.e. 15-30 fps) on mobile devices with low compute power. These models are still relatively good generalizing while keeping their runtime low.

Specialized models When trained for a highly specific problem, compressed models have proven to achieve near reference accuracy while running orders of magnitude faster. These models are specialized to the objects and objects' perspective of each deployment and do not generalize well to new scenarios.

Each category provides a different trade-off in terms of accuracy, runtime, and generalization. The justification for considering all three categories in our study is duly justified in previous work that shows how a combination of these categories can help to dramatically reduce runtime and costs while minimizing the accuracy loss [55, 7, 43, 42]. In summary, previous work in the field of video analytics has focused on maximizing the accuracy of Edge and Specialized models while minimizing the number of times the reference models (also known as *ground-truth models*) are executed. Only in cases of uncertainty, the reference model is invoked. In the context of a heterogeneous Edge, considering some or all of these categories is even more justified as each one may adapt better to a different platform within the edge infrastructure.

Table 3.3 provides more details about the models selected for evaluation on each of the three aforementioned categories. The selected models are a set of representative state-of-the-art models from the PyTorch Model Zoo [72]. The models vary in complexity and size. As previously mentioned, several previous works have focused their efforts on reducing the number of times an expensive, generalist model is required. Some works focus on creating a classification cascade where only in cases of uncertainty the reference model is needed [55, 7]. These works make use of highly inexpensive but also specific CNN. Other approaches propose to divide labeling in two steps: *ingest time* and *query time*, i.e. the moment in which live video is captured vs the moment at which a certain footage is queried looking for certain objects in scene [43]. This method executes a *cheap* NN during ingest time while more expensive NNs are reserved for when high accuracy is needed at query time. They compare the relative recall of ResNet18 and two compressed and specialized variants (with 4 and 6 layers removed and scaled to 112 and 56 pixels, respectively). The results of that work demonstrate how by considering the *top-K* instead of only top-1, they reach $\geq 99\%$ relative recall with respect to their reference model (YOLOv2 [74]). For the specialized models, we consider ResNet16 and ResNet14, i.e. a ResNet18 with 2 and 4 layers removed.

3.2.3 Re-thinking FPGAs for the Edge

Video analytics is an excellent match for shared accelerators. GPUs, for example, have been long prototyped and used in shared configurations both in the literature [29] and in production systems [5]. However, other accelerators, such as FPGAs, also show great potential while their adoption is still minimal.

One of the key benefits that make FPGAs so appealing is their capability of providing highly customizable hardware designs at a relatively low cost compared to regular ASICs. Moreover, once a bitstream is generated, FPGAs can be reprogrammed in a matter of seconds. However, their usage also implies a series of drawbacks pushing their adoption back. A few seconds to reprogram is negligible when it is required only once in a while but becomes prohibitive if we aim to have multiple tenants time-sharing the accelerators.

Moreover, the software stack and development kits lag behind with respect to those for software development, which makes the development costs for FPGAs orders of magnitude higher than those for other well-stablished accelerators, such as GPUs. Nonetheless, there have been some important advances at easing development [68] and OpenVINO, for example, supports FPGAs and can be downloaded with various pre-compiled bitstreams ready to accelerate multiple neural network architectures out-of-the-box. Easier development combined with FPGAs' claimed *high-throughput, low-latency, and low-power* [47], may finally make FPGAs a viable option to accelerate edge workloads.

When considering accelerated video analytics, FPGAs become especially appealing for several reasons.

- Requests are independent of one another, i.e., CNNs have no memory, and frames are processed independently, deeming expensive context-switching operations unnecessary.

- Video analytics relies on the execution of neural networks (usually convolutional neural networks or other classes of neural networks with similar execution characteristics). The same CNN architecture can be used by multiple models trained to solve different problems, which makes a single bitstream suitable to provide multiple services without re-programming the FPGA. Moreover, methods like the one we presented in [34] can be used to monitor the performance of multiple kernels co-existing within a single bitstream, in order to guarantee SLAs are met when the FPGA is shared among different applications.

- Even if different requests require different trained models, multiple of these can be stored together in the FPGA's main memory. There are existing technologies to allow the model's parameters to be sent along with the request and be stored in the FPGA's main memory without storing it first into the host's memory, like the one we proposed in [35].

Known Limitations

In Section 3.4.3, we provide the evaluation results video analytics accelerated by an FPGA. However, there are certain limitations that are important to consider when evaluating FPGAs in their current state as general-purpose accelerators and comparing them to other available options, such as GPUs or even CPU-only platforms.

First, the limited range of available options. FPGAs have been long used for the prototyping of ASICs and there are various manufacturers with a wide range of platforms for such purposes. However, large-scale and heterogeneous edge deployments demand FPGAs to be seen and used as general-purpose accelerators, but that is a relatively new concept. Consequently, once we limit the FPGAs to consider to those supported by the OpenVINO toolkit, the number of available options is rather limited. At the time of conducting this research, there was a single platform supported: the Intel Arria 10 FPGA [49] (although in different configurations and package options). Specifically, we have used an Intel Arria 10 GX, first launched in 2013 by Altera but re-factored in 2017 by Intel to be the flagship of their FPGA solutions for artificial intelligence. It is a passively-cooled and single-slot PCIe accelerator with 8 GB of DDR4 main memory, drawing up to 70 watts of TDP, close to the maximum the PCIe connector can provide (75 watts). On paper, it lags behind what other contemporary accelerators provided. On the other hand, the Arria 10 FPGA can deliver up to 40 Gigabit Ethernet thanks to its integrated network interface, while also containing specific hardware for digital signal processing that can be highly valuable in edge environments to accelerate telecommunications.

Second, FPGAs are constrained to what the bitstream provides. On the one hand, the capabilities of an FPGA are tightly coupled to the bitstream used. While obvious, it is essential to highlight that an FPGA will not be able to execute any operation that is not supported by the bitstream with which it was programmed. In general-purpose scenarios, we do not consider the possibility of customizing bitstreams on a per-application basis, due to the high development time each one would require. Therefore, the evaluation is limited to those models supported by the bistreams provided by OpenVINO. On the other hand, performance is also highly dependent of the bitstream. This makes it challenging to provide maximum throughput when two applications share a single FPGA with a single bitstream. Unless both applications require the same set of operations, the bitstream used may be the result of a trade-off between the space available and a compromised performance for the sake of finding a feasible solution.

Third, the FPGA does not support hardware decoding, which implies that the CPU cannot be completely freed. This is related to the previous limitation, as decoding support can be added through a new bitstream. However, adding decoding would mean giving up on neural network execution, which is unreasonable in the context of video analytics applications. Consequently, we evaluated FPGAs considering only their inference performance.

For all the reasons mentioned above, we have evaluated the FPGA separately from the other platforms.

3.3 Methodology

3.3.1 End-to-End Characterization

In this chapter, we evaluate the different trade-offs present in continuous video analytics. As presented in Section 2.2.2, the typical end-to-end video analytics pipeline consists of two major building blocks: frame decoding, and the execution of neural networks (i.e., inference).

We characterize both phases individually to, finally, show how considering the end-to-end pipeline impacts the selection of platforms in different environments.

Video Decoding

To evaluate decoding, there are multiple multimedia frameworks that are widely used and equally valid. Throughout this analysis, all decoding results were obtained using the *GStreamer* framekwork. GStreamer is a pipeline-based multimedia framework that allows to link a wide variety of media processing systems and build a highly customised workflow. OpenCV's VideoCapture class allows us to select GStreamer as the media backend and then specify the pipeline according to the format and the accelerator used.

Regarding the video formats, we considered three of the most widely used: H.264, H.265, and VP9. VP9 is rarely used in live-streaming applications but it is widely used in different Video On Demand (VOD) platforms, such as YouTube, and is open-source and royalty-free. On the other hand, H.264 and its more efficient evolution H.265 are proprietary and their use requires a royalty. However, these two are almost the *de-facto* standard when it comes to video live streaming and most devices world-wide (including IPTV cameras) include hardware support for H.264 and H.265.

Comparing video formats is not straightforward. There are many criterion by which video formats may be compared: image quality, compression ratio, speed, hardware support (or lack of). Moreover, encoders are configurable with multiple options to tweak all the aforementioned criterion. For this study, we wanted to focus the comparison on the performance for equivalent qualities, regardless of other encoding settings. Therefore, we have used a video downloaded from a video-on-demand platform and is pre-encoded in several formats and resolutions to match a wide range of user devices and are all expected to have, at least, similar perceived quality [103].

Neural Network Inference

The inference of neural networks has been evaluated using the OpenVINO toolkit for the CPU and MyriadX executions and PyTorch for the NVIDIA GPU. There are multiple frameworks available that are able to do training and execution and each one provides a different set of functionalities and optimizations. OpenVINO only supports the execution and not training, but it is the only framework that supports the Intel MyriadX accelerator. Nevertheless, in our evaluations we do not consider the training of the models and focus only on the inference. All the pre-trained models used in this chapter are publicly available in the PyTorch Model Zoo [72] and the OpenVINO Model Zoo [67] repositories. We used single precision models for the CPU and GPU evaluations, and half precision for the Intel MyriadX.

3.3.2 Evaluation metrics

Each of the phases of video analytics are evaluated under the following metrics:

Throughput Measured in frames per second (decoded or processed by the neural network).

Latency Time to process a single frame or request. We always consider the end-to-end latency and not the average latency, i.e. if a NN takes 2 milliseconds to process a batch of 2 images, we consider latency to be 2 milliseconds for each image and not the 1 millisecond average.

Power Efficiency Frames per second per unit of energy (Joules). For power consumption, we measure the average nominal power consumed during the execution interval. Then, we subtract the portion of the idle power that we believe should not count as part of the execution's power. For example, in a system with an Intel Xeon 4114 with a RTX2080, the idle power drawn is 25W for the CPU and DRAM, and 16W for the GPU. If we are measuring decoding on a single CPU core, we fully subtract the idle power from the GPU plus 22.5W from the CPU, as each one of the 10 cores adds 2.5W. If we are measuring inference on the GPU while using a single core, 22.5W are deducted from the CPU idle power while nothing from the GPU, as it is being fully utilized by the inference execution alone.

However, we do not consider the accuracy of the models nor the task for which they were trained for two reasons. First, the accuracy of a model highly depends on the training process and is extremely domain-dependent. Hence, we assume a model is selected upon its known accuracy and ability to generalize, and we focus on the trade-offs that such model may expose in different hardware platforms. Second, the execution graph of CNNs is usually fixed. Thus, regardless of a model's accuracy on a specific task, its throughput is determined by the type and number of layers (complexity) and the number of trained weights (size). Therefore, the conclusions of this study can be safely extended to equivalent CNN architectures trained on other tasks.

Environment, Tools and Platform Telemetry

All the platforms used for evaluation share the same setup: CentOS 7.8 (kernel version 5.4.15), with Hyper-Threading disabled and the *intel_pstate* CPU scaling governor, running Docker version 19.03.11. The platform with the NVIDIA GPU uses CUDA version 11.1 and driver version 455.23.05.

The telemetry of the hardware platforms was obtained using a custom Python package developed for this project [41]. Through different interfaces, the package collects multiple metrics to provide a global view of the system: system utilization for CPU, memory, disks, network, and sensors; memory bandwidth and cache metrics through performance monitoring counters; package, core, and DRAM power consumption through Running Average Power Limit (RAPL) interface for desktop and mobile architectures and through IPMI and performance counters for server architectures; GPUs are monitored through queries to the *nvidia-smi* command-line tool, or RAPL for the integrated GPUs. The components have some overlap regarding the statistics they provide, as not all platforms support all interfaces. Therefore, for each platform we used a combination of components to ensure a complete analysis.

3.3.3 Reproducibility of the results

One of the outcomes of this study is that the optimal platform will highly depend on numerous factors. Among these factors, some will be subject to change throughout time while some others will be pre-determined and fixed by the specifics of the deployment. Therefore, we wanted to make possible for anyone to reproduce this kind of study under different configurations.

In the public repository of this project [41], we provide all that is required to ensure the reproducibility of our experiments. The repository contains the scripts (for setup and execution) and the Dockerfiles to build the images we used for this chapter.

3.4 Results

In this section, we present the characterization of video analytics and use the cornerstones of its results to build the experimental model that will allow us to explore performance and cost trade-offs of different hardware configurations. The experiments are divided as follows:

1. Video decoding: characterization of the decoding phase. The analysis includes performance scalability, latency impact, and energy efficiency with respect to the resources allocated to the task, the encoding, and the resolution of the input video. Trade-offs of hardware media decoders are also explored.

2. Neural Network Execution: characterization of the inference phase. The analysis also includes performance scalability, latency impact, and energy efficiency with respect to the resources allocated to inference. Trade-offs of low-power (MyriadX) and high-end (GPU) neural network accelerators are also explored.

3. End-to-End Performance Projection. We build an experimental model from the findings of the previous characterizations to estimate throughput, latency, and power within different constrained scenarios. The experiments explore the trade-offs of the different hardware platforms and highlight how each one fits into a different context that the model can help identify.

3.4.1 Frame Decoding

When evaluating video analytics, neural network execution tends to get all the attention, and frame decoding is often overlooked. However, it is important that the execution of these two phases is balanced. In this section we start looking at frame decoding in isolation.

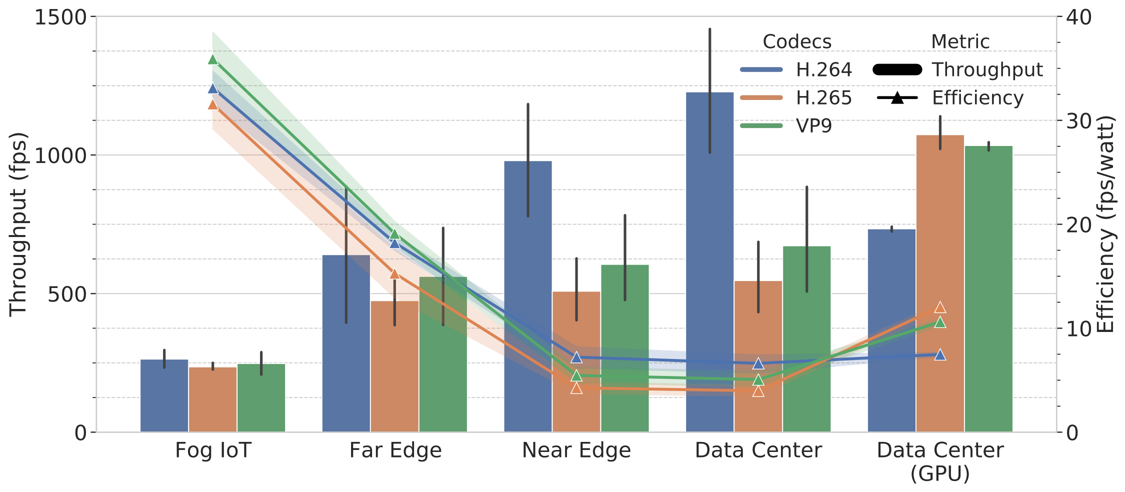


Figure 3.1: Decoding throughput, as number of frames decoded per second, (bars, left Y-Axis) and power efficiency, as frames per watt, (lines, right Y-Axis). All input videos with a resolution of 1920x1080 (FullHD).

Figure 3.1 shows a general overview of the decoding performance and the power efficiency of the analyzed hardware platforms. Far Edge, Fog IoT and Data Center with GPU platforms use the hardware decoder, while the Near Edge and Data Center platforms use only the CPU. From these results, we can extract two conclusions. First, the performance of each codec seems to highly depend on the actual implementation of the decoder. While CPU implementation of the H.264 decoder is 1.5x to 2x faster than its modern version H.265, this difference reverses on the NVIDIA GPU and vanishes on the Far Edge and Fog IoT, also using hardware decoder. On the other hand, VP9 seems to be consistently in between along platforms.

Second, we can observe how the more resource-constrained Fog IoT platform achieves the lowest performance but, at the same time, it is the most power efficient platform for decoding. This highlights one main differences among these platforms: user consolidation versus energy efficiency. Similarly, user aggregation is further emphasized in the server platforms. Near Edge and Data Center platforms are close in terms of both performance and efficiency, specially after adjusting to core count. However, it is important to notice that in order to keep the figure legibly and not distort the Y-Axis, these results correspond to a single socket of the Data Center platform, i.e., only showing half of its peak performance, although efficiency should not vary. Moreover, we observe how the performance of the discrete GPU is on par to what the other platforms achieve. This parity will prove to be particularly important when jointly considering decoding and neural network inference.

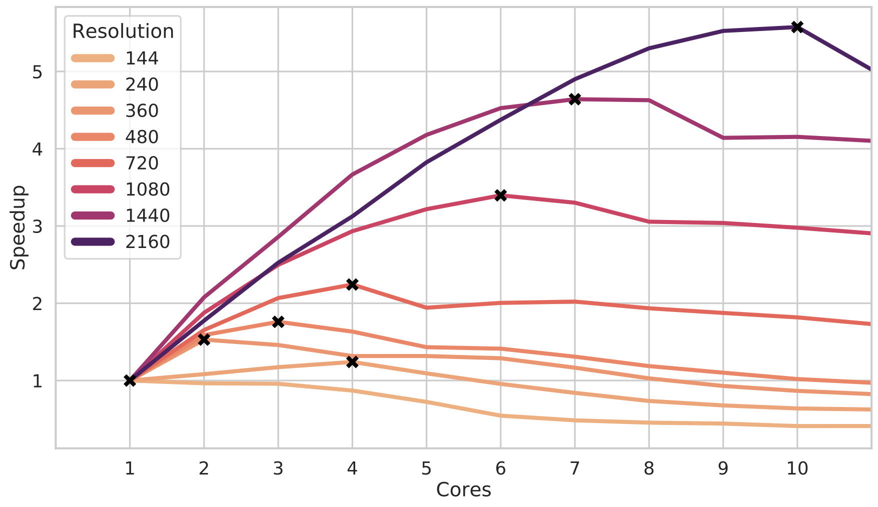


Figure 3.2: Decoding scalability. Speedup with respect to the number of cores used for decoding for different video resolutions. The x on each line marks the maximum speedup for the given resolution. From that point, performance decreases as more cores are used for decoding. All videos use VP9 compression. Experiments run on the Data Center platform using only CPU (Intel Xeon Gold 6248).

Scalability

Figure 3.2 shows how decoding performance scales with respect to the number of cores used for different input resolutions. The results correspond to single-process executions in which a single input video is decoded as fast as possible. Results show a clear impact of the input resolution on the scalability. The higher the resolution, the higher the number of cores needed to achieve peak performance. While a single core suffices to decode a 144p video at full speed, 6 cores are needed on a 1080p video and 10 for a 4K video. After reaching peak performance, increasing the number of cores degrades performance. We have observed this to be consistent among the codecs we considered.

Figure 3.3 shows the decoding performance and the average latency per frame as the number of cores increases. In many cases, decoding performance will not become the major bottleneck. Video streams at 1080p and 30 or less frames per second are common in video

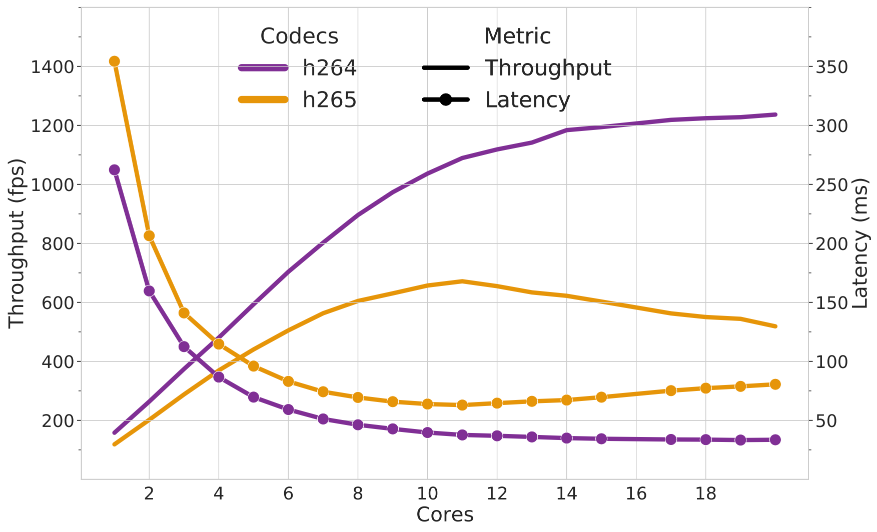


Figure 3.3: Decoding throughput (left Y-Axis) and decoding latency per frame (right Y-Axis) while increasing the number of cores used for a single process. Input: H.264 1080x1920, @25fp. All experiments were run on the Data Center platform (Intel Xeon Gold 6248).

analytics and the performance for a single core is more than capable of such frame rate. However, the number of cores used for decoding also impacts the latency per frame. As the figure shows, average latency drops around 40% for both H.264 and H.265 when moving from one to two cores. Using 8 cores for the decoding reduces the latency by an 80% with respect to the single-core execution. In some latency-critic applications, this may have an inevitably adverse effect on the end-to-end latency.

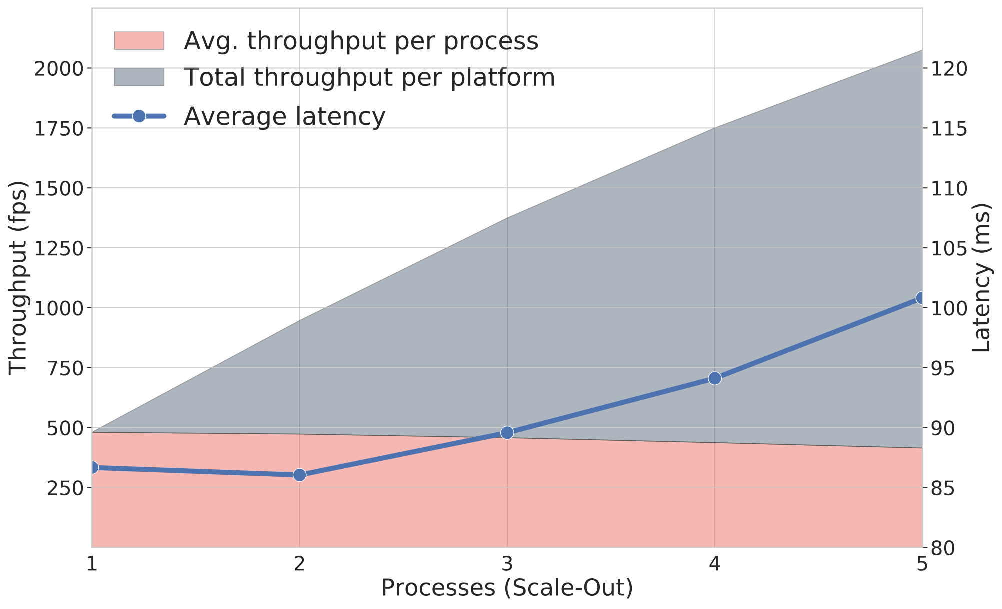


Figure 3.4: Decoding throughput (left Y-Axis) and decoding latency per frame (right Y-Axis) while increasing the number of processes decoding simultaneously and each process uses 4 cores. Input: H.264 1080x1920, @25fps. All experiments were run on the Data Center platform (Intel Xeon Gold 6248).

These results show that the decoding scalability benefits from more data. This is the classic problem of weak versus strong scaling. This particular implementation provides good

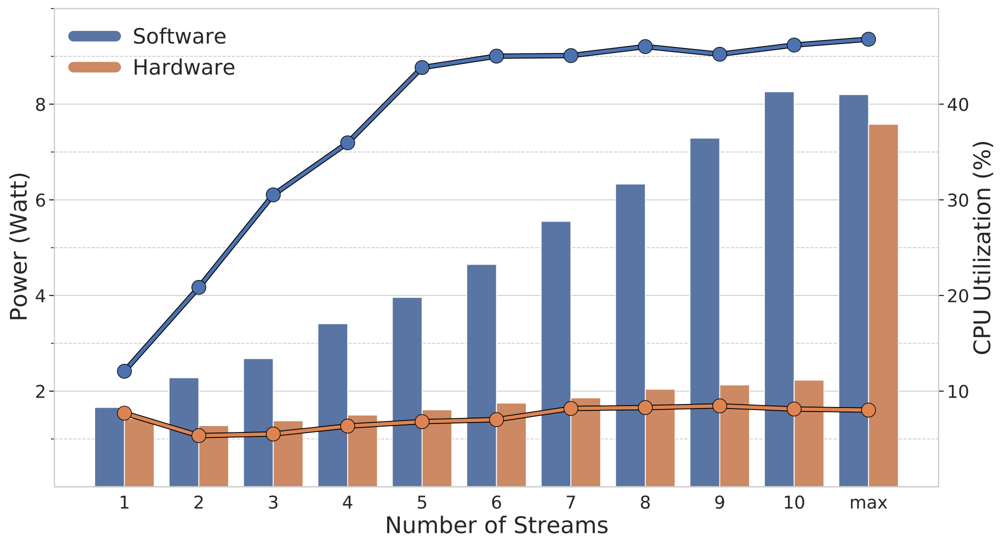


Figure 3.5: Power consumption (left Y-Axis) and CPU utilization (right Y-Axis) while increasing number of streams decoded simultaneously. All streams are in H.264 with a resolution of 1080x1920 at 25 frames/second.

scalability as the problem grows. However, for small data sets, i.e., images with small resolution, the application is not capable to scale with the number of cores. Therefore, we repeated the same experiment but increasing the number of processes while the number of cores remains the same to see how scaling out affects the decoding performance. Figure 3.4 shows the results of this experiment. The figure shows the average throughput per process and the total throughput accumulated by all the processes decoding simultaneously. The figure also shows the average latency per process. We can observe how the total throughput of the platform scales linearly, in contrast to the scale-up results. By using the 20 cores in the machine divided in 5 processes, i.e 4 cores per process, instead of a single one, we achieve a 4x speedup. The average performance per process drops by a 15% and the average latency increases by the same number. However, it is important to recall that the maximum frequency when all cores are working is reduced by a 20% (and a 15% with respect to 4 cores).

Software vs Hardware Decoding

The following experiment compares performance, latency, and energy efficiency of the software decoder and the hardware decoder (i.e., present within the integrated graphics card). Hardware accelerators, as previously mentioned, usually provide higher efficiency and/or higher throughput but tend to require full utilization to achieve them. In this experiment, we wanted to test both scenarios: low and high load.

Figure 3.5 shows the efficiency (in terms of average millijoules per frame), and CPU utilization while increasing the number of input streams (each stream is H.264 1080x1920, @25fps) decoded simultaneously and synchronized with the input frame rate. The load is increased from one to ten streams and, then, one execution with a single stream with asynchronous decoding, i.e., no limit on frame rate, to compare their peak performance. The hardware decoder yields up to 3.7x higher efficiency than the CPU at the same throughput and 4.2x higher efficiency when the accelerator is fully utilized. Moreover, the hardware decoder frees the CPU for other uses (e.g. DNN execution) and, in the experiments, does not go beyond 8% CPU utilization. We can conclude that the hardware decoder, contrary to other hardware accelerators, is more efficient than its software counterpart regardless of the amount of work. However, the main drawback of the hardware decoder is its flexibility, as the formats and format options it supports are fixed and limited, which may make it unfeasible in some scenarios.

3.4.2 Neural Network Execution

Figure 3.6 shows the throughput (in frames per second), the average latency per inference (in milliseconds), and the efficiency of the platform (in frames per watt). We divide the results into the three categories of models we described in Section 3.2.2. The first thing we notice is the clear difference among the three categories for all platforms and metrics. There is roughly one order of magnitude difference in performance for each category. With a closer look to the reference models, we observe that Faster R-CNN is itself one order of magnitude slower than ResNet152, achieving a maximum 15fps on the GPU. On the other end, we observe how the GPU achieves more than 25000fps on ResNet14, the smallest of the evaluated models. This makes the GPU 4x faster than the rest of the platforms. However, this is a number that will be revised again in Section 3.4.4.

Regarding the CPU executions, the Intel Xeon 6248 on the Data Center platform is well above the rest on six five models, even after averaging per core. The Intel Xeon D-2183IT on the Near Edge platform comes behind, as expected, with lower memory bandwidth (4x versus 6x memory channels) and lower boost frequency (3GHz vs 2.2GHz when utilizing all cores). On the other hand, the Intel i7-8700T on the Far Edge platform is relatively close to the Near Edge and this is due to one of the many trade-offs in edge deployments: user aggregation. The Far Edge's higher frequency (up to 4GHz) outweighs the bigger caches and higher memory bandwidth on the Near Edge platform server, as the performance of NN execution is tightly coupled to the core's frequency. However, the 16 cores of the Xeon ensure greater scalability to aggregate more users, along with an on-par 10Gbps Ethernet that desktop platforms lack. Finally, the so-called Fog IoT and the MyriadX accelerator (both considered low-power solutions) are left behind in terms of performance but on par between them, with the accelerator yielding slightly better throughput and efficiency. Nevertheless, the MyriadX accelerator has the advantage of freeing the CPU for other tasks.

Scalability

Similar to other parallel tasks, the amount of resources put into neural network execution may scale in two directions: up (i.e., more resources to process the same amount of work) or out (i.e., more resources to process more work in parallel). In practice, scale-up translates to requests processed faster (lower latency) and scale-out translates to higher throughput. Figure 3.7 shows the speedup of the different neural networks run on each of the analyzed platforms with respect to the compute resources scaled in scale-out mode. The baseline corresponds to an execution using a single core. Solid lines represent the speedup as it was measured by the application. We can observe how the data center and near edge platforms start diverging from their ideal speedup as the number of cores increase. While this may seem like a different bottleneck is hit (e.g. memory), it is important to notice that data center platforms' design is not driven by single-thread performance but by user aggregation and reliability. Therefore, due to thermal and power limitations, the maximum frequency at which a core is able to run is capped as more cores are used. For example, up to two cores in the data center platform are able to run at 3.9GHz simultaneously, but the maximum frequency is gradually reduced down to 3.2GHz when all 20 physical cores are being used. The dotted lines in Figure 3.7 represent the speedup adjusted to the maximum frequency,

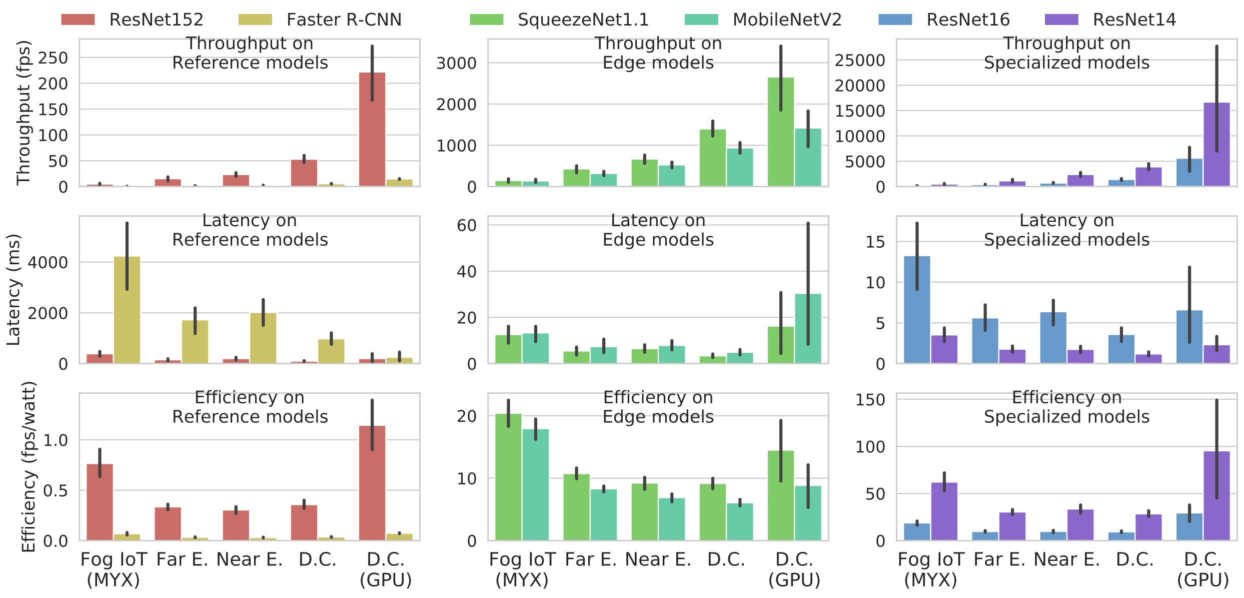


Figure 3.6: Throughput, latency and efficiency (fps per watt) comparison. It is important to notice that the Y-Axis is not shared among plots. (*acronyms*: MYX - Intel Myriad X; D.C. - Data Center)

i.e., frequency with a single core. It can be observed how this adjustment brings speedup back to near-ideal speedup.

On the other hand, with scale-up mode we have the option to increase the number of cores used for each individual request. Figure 3.8 shows the speedup of the different models on each platform as the number of cores increases. Notice that in scale-up mode, the latency of each request is equal to the inverse of the throughput, as the benchmark serializes the work. We can observe how the throughput scales very differently for each model. By taking a new look at the characteristics of each model, we can see how the bigger a model is (in terms of complexity and number of parameters), the more it benefits from more parallelism.

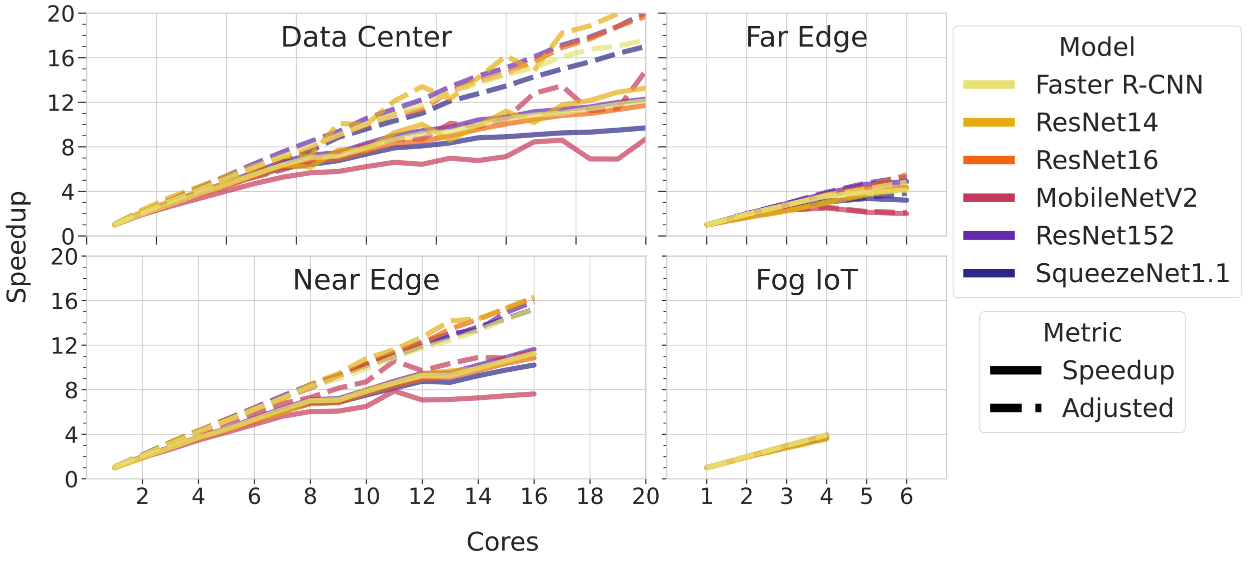


Figure 3.7: Throughput scalability with respect to the number of cores used for DNN execution on the different platforms.

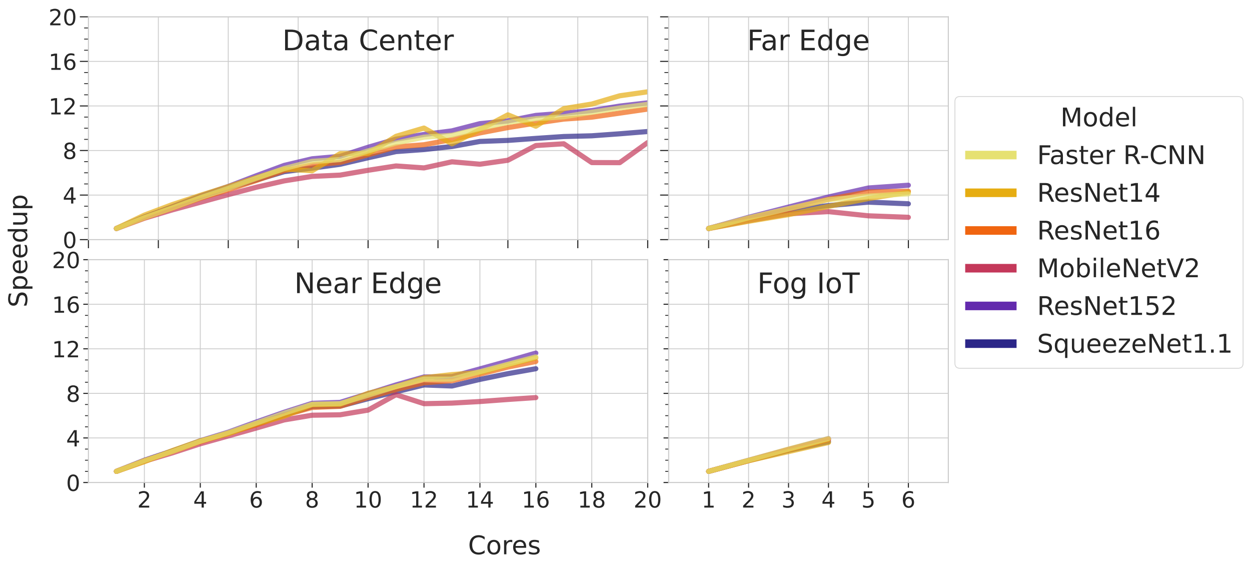


Figure 3.8: Throughput scalability with respect to the number of cores used for the execution of neural networks in scale-up mode, i.e., no parallelization of requests.

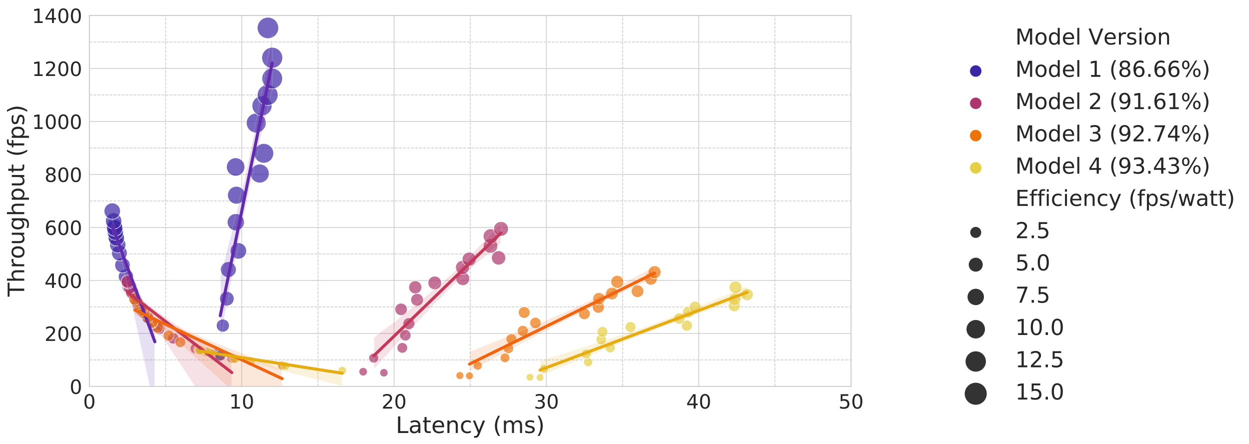


Figure 3.9: Inferences per second (Y-Axis), inference latency (X-Axis) and power consumption (bubble size) of face detection models executed on an Intel Xeon D-2183IT. All four models are based on MobileNetV2 as their backbone. Models 1 to 3 use 2x SSD (Single Shot Detector) heads as detectors. The higher accuracy is achieved by increasing the input size: 3x256x256, 3x384x384, and 3x448x448 for 1, 2, and 3, respectively. Model 4 achieves higher accuracy by using a FCOS (Fully Connected One-Shot) head, a more complex but more accurate detector. Model 4's input size is 3x416x416.

Accuracy

Depending on the use case, the accuracy of the model to deploy may be even more important than the number of inferences per second it can achieve on a specific hardware. As already mentioned, we do not consider accuracy as another variable on our search space as we assume it highly depends on the task at hand while the acceptable minimum to deploy the model should be decided during the specification of the use case and guaranteed during training. Nonetheless, accuracy of a given neural network architecture can be enhanced through various means, such as using different operations, adding layers or increasing the input size. Therefore, we would like to understand how a higher accuracy may impact throughput and latency. The experiment depicted in Figure 3.9 considers four detection models using the

Model	**AVX-512 (+VNNI)**	**AVX-512**	**AVX2**
YOLOv2	235.8%	84.9%	66.6%
Faster R-CNN	178.9%	82.4%	55.0%
MobileNetv2 + SSD	149.9%	47.3%	9.2%
Average	188.2%	71.5%	43.6%

Table 3.4: Speedup achieved by moving from FP32 to INT8 precision on CPU.

same backbone model (MobileNetV2) in four different flavors with different accuracies. The figure shows the number of inferences per second, the inference latency, and the power consumption of face detection executed on the Near Edge platform. The different data points correspond to executions with different number of threads per inference and different number of simultaneously executed inferences. The data points closer to the origin correspond to scale-up executions and the others correspond to scale-out executions. We can observe how each model has its clear distinct space in the grid. Accuracy (working as a proxy for complexity in this scenario) determines the scalability of the model. Moreover, accuracy expands or shrinks the area in which resources have to be allocated to meet constrained scenarios. The regression lines of each model further emphasize what the scalability experiments showed: scale-out increases performance linearly while scale-up reduces latency logistically.

Apart from the input size, quantization is another technique that aims to improve inference throughput. Quantization reduces the size of each weight in the model. Therefore, improving the model's density (weights per bit), specially important with accelerators and vector units. If done properly, the drop in accuracy caused by the lower precision is minimal. For the following experiment, we used a different set of models available in the OpenVINO Model Zoo [67] already quantized in INT8 precision. To quantify the speedup brought by the processor's vector unit, and more interestingly, the new VNNI instruction set, we compare the performance gains on three of the platforms. It is important to recall that the VNNI instruction set only works with 8-bit integer operations. We use the performance of the same model with single precision (FP32) as baseline for each of the three CPU platforms and then compare the performance gains by moving to 8-bit weights. Table 3.4 summarizes the results. The Intel Xeon Gold 6248 on the Data Center platform (with support for VNNI) achieves 2-3x higher speedup than an Intel Xeon D-2183IT, both with AVX-512.

3.4.3 FPGAs: an alternative to accelerate the edge?

Figure 3.10 shows the performance of the FPGA compared to that of the CPU on the same platform. The platform is the same Data Center platform that has been paired with the GPU in previous experiments, i.e., an Intel Xeon Silver 4114. It is important to highlight that this platform has not been used in previous experiments in a CPU-only configuration because of its low base and boost frequency when working in AVX-512 (which OpenVINO requires) of 1.1GHz base frequency, which implies a 50% reduction compared to the 2.2GHz base frequency in normal mode. Yet, the FPGA, albeit being a 70-watt accelerator, does not seem to be on-par even with a mid-range server CPU. While the FPGA does surpass the CPU in terms of throughput on some of the models, this difference does not seem to pay off the overwhelming advantage of the CPU on other models.

On the other hand, one advantage of FPGAs is that they serialize requests. Therefore, a request's latency is always the inverse of the throughput and has a lower variability than the CPU. The CPU's throughput and latency depend on the number of cores used and whether the cores work all together on a single request (latency mode) or each on a separate request (throughput mode). The small variability shown in Figure 3.10 is due to the time taken to preprocess the input of the request. Nonetheless, these results hint that the performance of the FPGA will vary from model to model, as different models require different operations whose implementation on the FPGA might not be optimal due to either design or space available on the board itself.

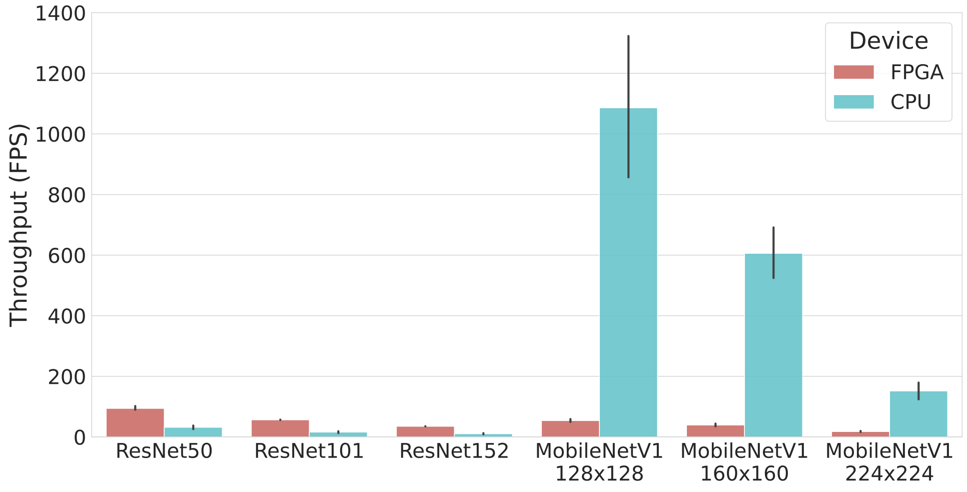


Figure 3.10: Comparison of FPGA and CPU-only throughput for different models.

Next, we tested how the choice of a bitsream over another affects the performance of the FPGA. Different bitstreams implement different operations and are optimized for different neural network architectures. Moreover, OpenVINO exploits the high level of customization offered by FPGAs by providing bitstreams with half precision (FP16) along with FP11, which would not be possible in other general-purpose hardware platforms. Figure 3.11 shows the performance of the Arria 10 when executing different models using different bitstreams. In this case, the model selection was limited to models that could be executed using different bitstreams from those available in OpenVINO, as unsupported operations move the execution back and forth to the CPU, which degrades performance to a point it is not realistic to consider it. On the one hand, these results highlight the importance of selecting the right bitstream to achieve the maximum throughput the FPGA can deliver. On the other hand, they demonstrate one of the challenges of sharing an FPGA among multiple applications, as the optimal bitstream for one application may not be optimal for another.

Finally, a note on the alleged high energy efficiency of FPGAs. We encountered an unexpected problem when connecting the FPGA to the node. The Arria 10, with a TDP of 70W, generated a considerable amount of heat that caused intermittent system reboots unless fans were set to constant maximum speed. Usually, the TDP of other accelerators also accounts for the power required to dissipate it as they leverage built-in fans connected to the accelerator's board. However, being passively cooled, the Arria 10 shifts this responsibility to the server itself. Consequently, a platform that normally draws around 115 watts when idle had to draw close to 250 watts as soon as we connected the FPGA. Figure 3.12 shows the power consumption of 5 hardware configurations when idle broken down by CPU, DRAM, and the difference between the sum of the previous and the measurement taken from the PDU. This difference entails the power consumption of various components (such as hard drives, PCIe or USB accelerators, network cards, and fans), whose power cannot be read from the platform

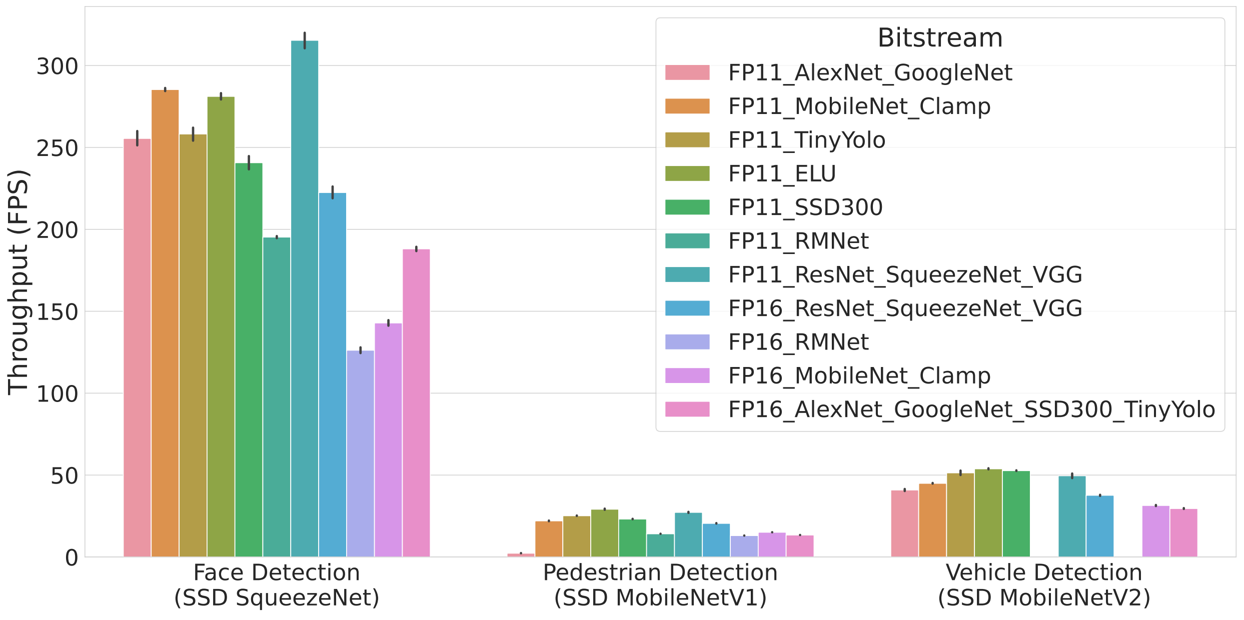


Figure 3.11: Comparison of FPGA throughput for different models when executed using different bistreams.

itself. For this experiment, we used the same Data Center (Gold), Near Edge, and Far Edge platforms as in previous sections, along with the Data Center (Silver) that we coupled with the FPGA. Figure 3.13 shows the same power measurements but in percentage of the total. These results highlight the unexpectedly high amount of power measured by the PDU when the FPGA is connected, accounting for a 84% of the total idle power. For comparison, the PDU measures a 61% difference on the same platform without the FPGA and a 70% on the high-end Data Center (Gold), which, at the time, had 6x NVMe of storage and 6x DIMMs of Intel Optane Memory drawing additional power.

These results explain part of the low energy efficiency we measured during the experiments. Nonetheless, the increment in power consumption expected when transitioning from idle to work at full speed is, indeed, reduced, as fans are already at their maximum speed and fans (or cooling mechanisms in general) are a considerable part of a server's power consumption. Unfortunately, we were unable to find a solution to this problem. As a result of this unresolved issue, together with the poor performance obtained in our experiments, we decided to leave the FPGA behind and deem it unsuitable for edge video analytics at the time of conducting the experiments.

3.4.4 End-to-End Performance Projection

Previous results have shown some of the key differences between CPUs, GPUs, and the MyriadX accelerator, as well as how decoding and inference scale with respect to the available resources.

Decoding There are three factors to take into account when evaluating decoding: video resolution, video codec, and whether hardware acceleration is available. The first two factors are rarely decided at will and are commonly determined by the kind of cameras that are installed. The third one can, or should, be influenced in cases in which decoding may become a bottleneck. Results have shown how the scalability of the software decoder increases as the resolution increases. However, there is a point at which more

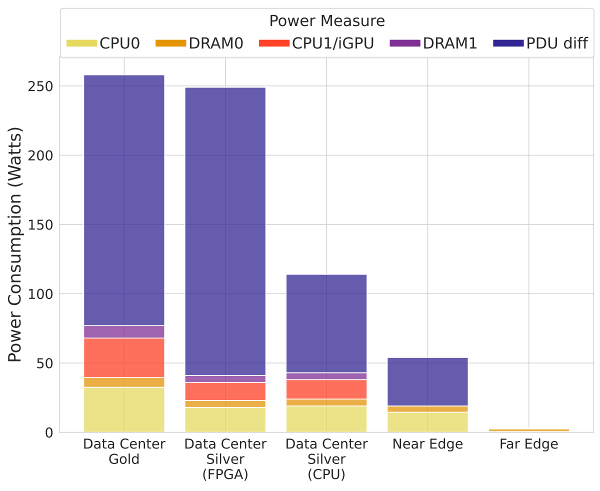


Figure 3.12: Breakdown (CPU, memory, and PDU) of the idle power in Watts of the power consumed by the platform.

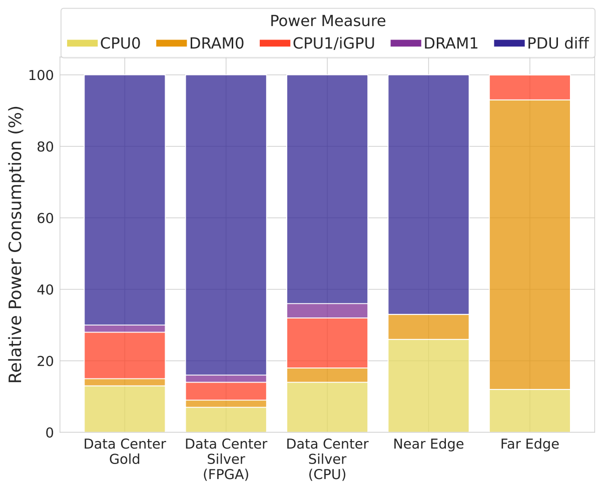


Figure 3.13: Breakdown (CPU, memory, and PDU) of the idle power relative to the total power consumed by the platform.

cores only increase the overhead and the decoding speed goes down. This is more important than it may seem because, in many applications, the decoding and the inference phases are not decoupled and use the same cpuset. A common pitfall is to increase number of cores hoping to increase inference speed while, in reality, the decoding is slowed down. This is specially important in a synchronous workflow, where the latency of each independent phase cannot be hidden by the previous phase and the end-to-end latency increases proportionally.

Inference CPUs are able to operate in two different modes: latency mode or scale-up (increasing the number of threads per inference) and throughput mode or scale-out (increasing the number of inferences executed simultaneously). In throughput mode, performance (inferences per second) increases linearly with respect to the number of cores at the expenses of higher latency. Higher latency is caused by frequency capping, which increases as more cores are used. However, performance scales linearly with respect to the number of cores for all reviewed models. After removing the effects of the reduced frequency, scaling is close to ideal in most cases. On the other hand, latency mode serializes requests to reduce the latency. However, the degree at which latency can be reduced will depend on the size of the model, as smaller models will see no benefit from latency mode. At the same time, the size of the model has an inverse effect running in throughput mode and bigger models will see an increment in latency as more requests are executed in parallel. Moreover, inference performance does not depend on the contents of the frame, and so when a neural network has been characterized in a platform, the performance is known (considering caches are already warm), and the latency of a given request is likewise invariant. Hence, performance and latency are affected by a set of deterministic factors, such as core frequency or number of cores available.

Finally, we combine both video decoding and execution of neural networks and consider the end-to-end pipeline to see how performance and cost per frame are impacted. Figure 3.14shows an estimation of throughput and cost for the different platforms evaluated in this chapter considering a FullHD video stream encoded in H.264. We focus on a single model for each of the three model categories used during the characterization. ResNet152 as the reference model, MobileNetV2 as the edge model, and a ResNet14 (ResNet18 with 4 layers removed) with its input size downscaled to 56x56. Then, we limit the maximum latency allowed for each configuration to be considered. The throughput (frames per second) corresponds to configurations that minimize the cost (watts per frame) and the cost corresponds to configurations that maximize throughput. As we relax the latency constraint, performance increases and cost reduces, especially for the ResNet152. We observe two major points. On the one hand, the execution of large models does require the use of a GPU, unless latency constraints are largely relaxed. On the other hand, we observe how in the case of small models, the GPU is relatively close to the CPU platforms, as decoding becomes the major bottleneck. Moreover, when considering the cost (in this case, the inverse of efficiency), resource constrained platforms even with low-power accelerators such as the MyriadX, are close to what throughput-oriented accelerators provide.

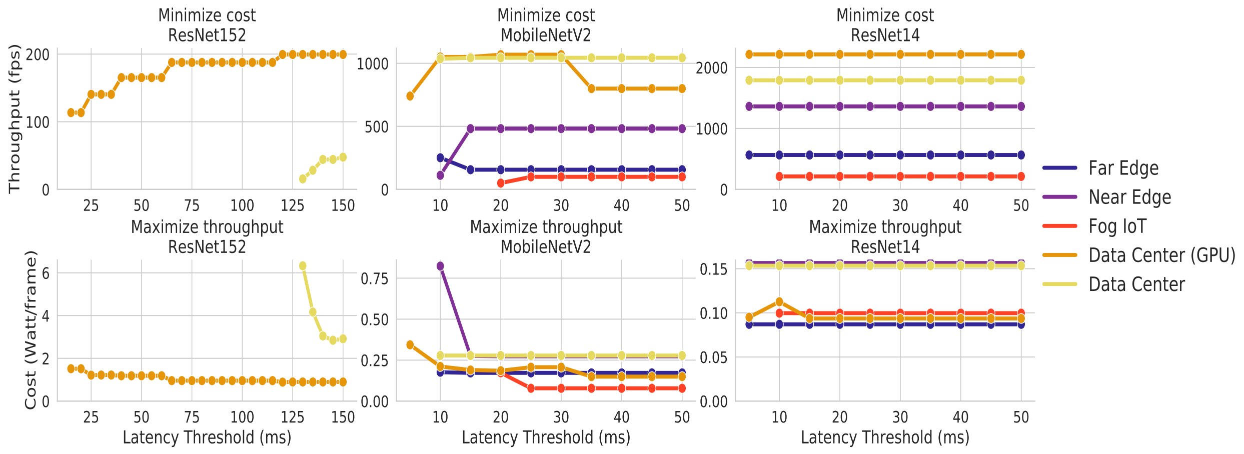


Figure 3.14: Estimation of end-to-end throughput minimizing cost and cost while maximizing throughput for one model on each category reference, edge, and specialized. Input video: H.264 and a resolution of 1080x1920 pixels.

3.5 Final Considerations

In this chapter, we have characterized a heterogeneous set of neural network architectures for video analytics using a heterogeneous set of hardware platforms, covering multiple types of edge deployments and use cases.

First, we have presented a taxonomy of edge locations (namely Far Edge, Near Edge, and Data Center), and how each of the selected hardware platforms map to these locations. Similarly, we have presented a taxonomy of neural networks (namely reference, edge, and specialized) that allowed us to consider neural networks' performance along with the relative accuracy we can expect from them. Both taxonomies are key to place platforms and neural networks within a context of edge deployments, where the hardware to deploy and the use case to provide are both tightly coupled to the restrictions imposed by the geographical region of the deployment itself.

Second, we presented an end-to-end characterization of real-time video analytics and analyzed its building blocks, video decoding and the execution of neural networks, separately. For this purpose, we evaluated a heterogeneous mix of hardware platforms, including three types of accelerators (a throughput-oriented NVIDIA GTX2080, low-power Intel MyriadX, and a discrete FPGA connected through PCIe). To the best of our knowledge this has been the first work to provide an end-to-end characterization of video analytics while also considering the hardware heterogeneity that is expected the edge to have.

Results have shown that decoding and inference are subject to different considerations and demonstrated the implications of evaluating platforms solely by their inference performance, as both (decoding and inference) can become the bottleneck, depending on the platform and the use case. These results demonstrated how each one these platforms is best suited for a different deployment.

Additionally, we have shown how, at the time of conducting the experiments, FPGAs are still a long way to accelerate general-purpose applications in the edge. While technology and development kits have considerably improved, the number of applications FPGAs can accelerate is still rather limited, while the number of available hardware options is even more limited. Moreover, according to our results, performance is not on-par with other contemporary accelerators, while orders of magnitude behind more modern (and more efficient) accelerators. Nonetheless, FPGAs provide high throughput and high efficiency on telecommunication workloads that must be present in any edge deployment. As such, FPGAs can still be useful in the edge, just not yet for video analytics applications.

Finally, we have introduced the key outcomes of the characterization into a projection model to estimate end-to-end performance, latency and cost for video analytics. Thanks to the model, we have been able to quantify the impact of considering the full end-to-end pipeline of video analytics and analyze platforms in terms of throughput and cost on different deployment scenarios. Moreover, it also allowed us to highlight the strengths and weaknesses of each platform for different types of deployments and show where each platform can be best deployed.

In closing, this chapter sets the ground for the upcoming contributions by defining the requirements of a heterogeneous edge that is expected to accelerate video analytics.

Chapter 4

Edge-Cloud Interplay for Large-Scale Edge Deployments

In Chapter 3, we presented a detailed characterization of video analytics, yelding the raw performance we can expect from different hardware platforms, including both processors and hardware accelerators. Through it, we have demonstrated that even the lowest-end processor or accelerator is able to provide end-to-end real-time performance with even higher efficiency than a high-end GPU, specially when the limited user aggregation does not allow to fully utilize it. However, we have also seen that such performance is highly dependant on the specific neural network being executed. As an example, Table 4.1 shows how an NVIDIA RTX 2080 can process 50x more inferences per second of a ResNet152 than a Myriad X. A single MyriadX cannot provide enough throughput for real-time applications, let alone latency, even if we were to stack several of them working together. However, a ResNet14 (used as a proxy of a highly specialized model) is a whole different story. The GPU can still execute 80x more inferences than a Myriad X but this difference, as overwhelming as it looks on paper, moves towards irrelevancy as we start considering the full end-to-end pipeline. To process 30,000 images per second the GPU needs 30,000 images per second (sic). 30,000 images that have to be decoded and transferred to the GPU every second. And the characterization of Chapter 3 has shown how this can become a problem rather easily as the decoding becomes the bottleneck (and the node still needs to aggregate that many requests from multiple users). On the contrary, 500 inferences per second is enough to provide real-time service to several users concurrently, all while consuming a handful of watts. Nonetheless, what can we do with such simple neural networks? After all, there is a reason why neural networks keep growing by stacking layers and adding parameters. As we limit the number of layers and parameters of a neural network, we are also limiting the

Device	**ResNet152**	**ResNet14 (Specialized)**
Myriad X	5 fps	501 fps
2080 RTX	269 fps	43,121 fps

Table 4.1: Inferences per second of a Myriad X and an NVIDIA 2080RTX accelerators for ResNet152 and a specialized model. Data extracted from the experiments in Chapter 3

.

complexity and the variety and robustness of the features it can learn from the training data. Therefore, by extension, we are limiting the accuracy they can achieve. On the other side, if we had means to narrow the scope of a problem down to simplify and reduce what the neural network has to learn, we could use specialized models without any loss of generality (i.e. accuracy).

The characterization on Chapter 3 focused on the raw performance while disregarding the use case for which the neural networks were trained as well as the accuracy the models achieved after being trained. Nonetheless, these two points are essential to deploy any meaningful video analytics application. Consequently, in this chapter, we focus on how to effectively use smaller and faster neural networks that resource-constrained platforms by specializing them to each deployment.

One downside of model specialization is that, as we narrow the scope of the problem down, we also narrow down the number of scenarios in which the model can be effectively deployed, potentially requiring one model per camera. For smaller and more specialized models it may even mean that different models are needed throughout the day as the same model may not be able to distinguish the same object with different lighting as the sun goes down. On top of that, each model implies one training process, while each training process requires its own accordingly sized and annotated set of examples. As the number of models grows, this quickly becomes unfeasible. For this reason, COVA leverages a hybrid Edge-Cloud architecture where the image capturing and contextual analysis happens in the Edge and the training is offloaded to the Cloud. Moreover, thanks to the computation power available at the Cloud, COVA annotates the training examples using a *ground-truth* model that works as an imperfect oracle.

In this chapter, we present COVA (Contextually Optimized Video Analytics), a framework to automate the specialization of models for real-time edge video analytics. Through it, COVA funnels the generality of big neural networks with state-of-the-art accuracy into specialized edge neural networks, resulting in models with high accuracy but on a narrowed scope. Our work is based on a simple observation: *static cameras do not move, but objects do.* At the same time, in the context of edge video analytics, we can safely assume that moving objects are equivalent to objects of interest, which allows us to leverage computer vision techniques to simplify the task. Moreover, we define the context of a camera as the set of invariant properties associated with its deployment location. For example, height or focal distance determine the perspective from which objects are seen. Similarly, the background and the environment (e.g., urban area, indoor, or nature) determine the type of objects that we can expect the camera to capture. Once the context of a camera is defined, automating the specialization of models can be vastly simplified.

The contributions of this chapter are two-fold. On one hand, we present and evaluate COVA (Contextually Optimized Video Analytics), a framework to assist in the rapid deployment of tailored edge models for real-time video analytics. The pipeline in COVA is conceived to be highly modular and easily extensible to allow for different types of contextual analysis of the scenes. It supports TensorFlow [92] (training and inference) and OpenVINO [69] (inference only) and can run either local or on *Amazon Web Services* (AWS) and orchestrated from the edge. Moreover, source code is made publicly available [77]. On the other hand, we present an extensive exploration and review of the different steps involved in the process of automating the specialization of models, including considerations and findings to be taken into account in case some of the pipeline stages are to be exchanged or extended.

This chapter is organized as follows. First, Section 4.1 revises the previous work and discusses the gaps that our work aims to fill. Then, Section 4.2 introduces how the context surrounding edge deployments can be exploited to specialize models to increase their accuracy without increasing their inference cost. Following, Section 4.3 breaks the problem of automating the specialization of models down, and provides the background to understand the challenges behind deploying accurate neural network models to the edge. Section 4.4 takes over the previously presented knowledge of the problem and presents the COVA framework as well as its components. Section 4.5 describes the methodology followed throughout the evaluation of COVA, which Section 4.6 presents in detail. Finally, Section 4.7 presents the final considerations of this chapter and discusses the results.

4.1 Related Work

4.1.1 Edge Video Analytics

The edge cloud has been previously proposed as a key enabler of video analytics [84]. In some cases, the edge is considered the *only* realistic approach to meet the latency requirements needed for real-time video analytics [6], while maximizing QoE by minimizing the cost and flow of data between data centers and users [17]. However, the edge cloud presents as many opportunities as new challenges that will have to be properly addressed on any new edge deployment [84, 54].

Within the context of edge video analytics, there have been lots of efforts focused on optimizing the data flow between edge devices and the data center. Authors in [4] propose an edge infrastructure with the pipeline for deep learning execution distributed across different points in the network (edge, in-transit, and cloud resources). Another optimization point consists of filtering what is sent to the data center. Authors in [22] present *FilterForward*, a system aimed to reduce bandwidth consumption and improve computational efficiency on constrained edge nodes by installing lightweight edge filters that backhaul only relevant video frames.

4.1.2 Model Specialization

Model specialization is recurrently used as a mean to optimize the inference cost in scenarios where generality is known to be unnecessary. On the one hand, some approaches leverage specialization over time through online training. For example, authors in [83] identify short-term skews in the class distribution often present in day-to-day videos and exploit it by training models specialized for such distribution online. On the other end of the spectrum, authors in [20] propose *Once-for-All*, a neural network architecture that can be repurposed and deployed under diverse architectural configurations but is trained only once.

Moreover, model specialization in edge video analytics is core to various video query systems [55, 43, 53, 45]. Such systems aim to answer queries about a video by analyzing its contents (e.g., return frames containing instances of a red car between two points in time). Authors in [55] are able to reduce the cost of running queries against a video by up to three orders of magnitude thanks to *NoScope*, a system that uses inference-optimized model search to find and train the optimal cascade of binary classifiers specialized for the video being queried. Authors in [43] propose the use of inexpensive specialized NN at ingest time (i.e., when images are captured) to filter the frames to be considered at query time. Nonetheless, these systems focus on the optimization of queries. Therefore, their main metric to optimize is the latency of a query (involving multiple inferences) and not that of a single inference. Consequently, such systems are subject to a different set of requirements and constraints than real-time edge video analytics.

4.1.3 Automatic Training

Model specialization at large-scale can only be feasible if the process is automated. When the goal is to generate specialized models, previous works have focused on distilling previously acquired knowledge from complex and accurate models (teachers) into simpler ones (students) [38]. Through distillation, the student model is trained by trying to mimic the output of the teacher model. On the one hand, distillation can be used to transfer all the knowledge of teacher model into the student to reduce its computational requirements while retaining *most* of the teacher's accuracy on the original dataset [70]. On the other hand, distillation can be used to generate specialized models in cases where the problem to be solved by the student is a subset of problem for which the teacher was trained. In such cases, the teacher model works as an oracle that is consulted to predict the labels of the training examples for the student [63, 55]. In this chapter, we focus on the latter to automatically specialize models for the context of each camera deployment.

Other works have focused on self-supervised learning. Authors in [39, 71] demonstrate that such methods are able to outperform traditional methods for supervised training in terms of accuracy. However, while very promising, these methods rely on vast amounts of compute resources and tackle the problem from a completely different baseline than does not fit the requirements and restrictions of the edge cloud.

4.1.4 Spatio-Temporal Locality

When analyzing video feeds from static cameras or any other kind of video where the camera remains still, there is a piece of information that is key to guessing where interesting objects may lie: anything new in the scene Vehicles, persons, birds, or anything moving is subject to be identified. Spatio-temporal information can tell us when something has moved within the range of view, and the region around the moving object can be used as the region of interest to start looking. Authors in [63] leverage the spatio-temporal locality captured by static cameras to drive the online training of a small specialized model, while authors in [101] propose a training mechanism in which temporal information is used to train a deep neural network at *instance level* (instead of pixel level) and successfully identify even occluded objects.

Beery et. al take this approach a little bit further with Context R-CNN [14] and propose a neural network that is divided into two stages: the first one is trained as a traditional R-CNN for object detection whose region proposals are used to train an LSTM. Stage 2 is an attention block that will consider the information of previous frames where detected objects have already appeared. The authors used video feeds from the span of more than one month to train the Context R-CNN. This method is able to correctly identify and detect objects even under challenging environmental conditions such as dense fog. However, authors in [13] make the observation that without sufficient data variability, models learn to focus on correlations in the background, which leads to poor generalization to new deployments. In this chapter, we propose to take the previous observation and invert the variable to optimize. Instead of finding methods to generalize to new deployments, we propose to specialize models for each deployment using images from the same static camera. By removing the generalization as the goal, simple and lightweight architectures for object detection become available, which is a requirement to run real-time video analytics in resource-constrained nodes in the edge.

This chapter presents and evaluates COVA, a framework to provide automatic specialization of models for real-time edge video analytics and tools to easily develop custom video analytics applications. Moreover, COVA is envisioned with extensibility in mind and could be used in conjunction with some of the methods mentioned above to further increase accuracy or speed. To the best of our knowledge, no previous work provides the framework and tools that COVA provides.

4.2 Exploiting the Contextual Edge

The edge cloud aims to provide compute and storage resources closer to where the data is produced to filter the amount of data crossing the backhaul and alleviate it. Nevertheless, as shown in Chapter 3, a geographically distributed edge cloud entails two characteristics whose implications are often undermined. First, different physical locations (at which resources are installed) involve different sets of constraints that ultimately impact the type and size of the resources that can be installed [80]. For example, a street cabinet might be limited to a handful of server-grade nodes, while a solar-powered node in a street pole might have to settle for an IoT gate with a low-power processor. Regularly, resources become scarcer the closer we are to the edge of the network. Consequently, the more challenging yet more critical it becomes to leverage those resources efficiently.

Second, the workloads (i.e., services) and the distribution of the incoming requests are tightly coupled to the geographical area the edge location is serving. This relation has long been experienced and exploited in traditional CDNs (Content Delivery Networks). Several factors, such as the time of the day or local trends, shape users' access patterns to the different internet locations daily. CDNs leverage this information aiming to predict what users will access next and cache it near them. Together, the combination of scarce resources, workloads coupled to a geographical location, and any other outside-world data that might affect the workload makes the context of the edge deployment up.

Video analytics, powered by all kinds of DNNs (Deep Neural Networks), is already one of the prominent use cases being executed at the edge [105]. At the same time, the edge has been appointed as a required actor and the primary accelerator for large-scale video analytics [6, 17]. All over the world, cameras generate constant data streams whose contents remain largely disregarded unless a specific event triggers their review. Some of these events require immediate action (e.g., alert security if someone breaks in a store or open a barrier after a car approaches it upon checking credentials), while others involve bulk analysis (e.g., crowd counting at a fair or car counting for traffic analysis in smart cities). However, in both cases, action and information are taken at a local level. Therefore, images can be processed in the edge while only a fraction of the data (the results) must be sent back to centralized locations to be registered.

Moreover, there is a direct relationship between the accuracy of a trained neural network and the size of the network (i.e., number of trainable parameters). Unfortunately, bigger networks tend to have higher computational complexity than simpler ones, as depicted in Figure 4.1. Ultimately, resource constraints on edge nodes set a limit to the size of the neural networks that execute and, thus, also to the accuracy they can achieve. State-of-the-art accuracy is usually out of the picture for the so-called *edge DNNs* and is only within reach of bigger neural networks. The difference between these two may imply orders of magnitude more trainable parameters, and the gap is only increasing with time [44]. At the same time, hardware acceleration becomes a must to provide anything near real-time performance. Altogether, these limitations pose a challenge for the edge cloud to postulate itself as a serious alternative to current data centers for video analytics unless the hardware installed is upgraded or the models deployed are greatly optimized.

One of the main reasons for bigger models to achieve higher accuracy is that the extra trainable parameters can be used to learn more robust high-level features that allow the model to generalize its predictions in new environments successfully. Simpler models may

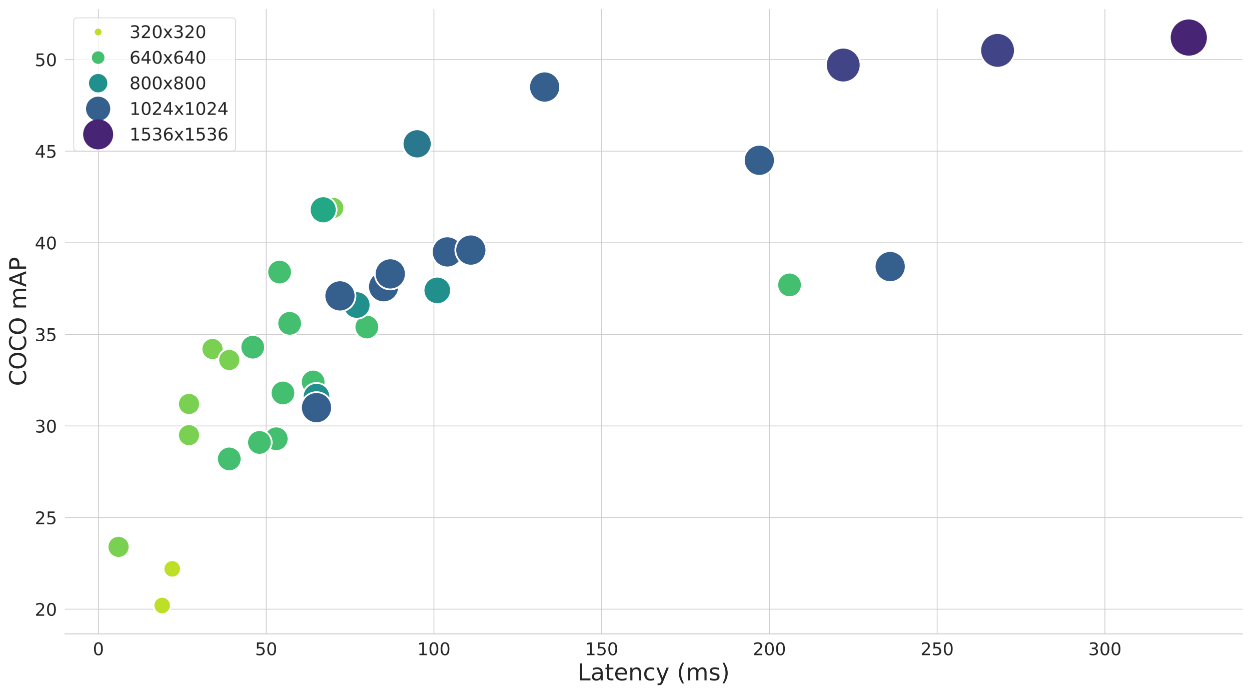


Figure 4.1: Speed versus mAP (mean Average Precision) on the COCO for different object detection pre-trained models [93]. Bubble size represents the input size of the model.

face difficulties in capturing the relationship between the input examples (i.e., images) and the target values (i.e., predictions), which is known as *underfitting*. However, it is essential to emphasize that simple or big are adjectives relative to the scope of the problem, and neural networks should be dimensioned according to the level of generality and complexity required for the task at hand. Otherwise, the *underfitting* we experience whenever we expect too much from a model may turn into *overfitting* when the network is big enough to memorize the training examples, which will translate into poor generalization on unseen examples.

Several popular challenges evaluate neural networks on specific datasets to tell how well a neural network architecture might perform. For example, ResNet152 scores 94.5% Top-1 accuracy on the ImageNet [46] dataset, while a smaller edge model like a MobileNetV2 is limited to *only* 71% Top-1 accuracy on the same challenge [16]. However, challenges like ImgeNet or COCO (Common Objects in Context) [60] are notoriously generalists, and models are required to distinguish a car from a toothbrush while also telling a *water snake* and a *vine snake* apart. Such diversity of objects is rarely captured by static cameras deployed to the edge. Just as a camera in the wild will hardly see many trucks, a camera controlling traffic at an urban intersection will not capture many wild animals. By narrowing down the diversity of objects to detect, simpler neural networks become available to be executed in the edge in real-time.

Nevertheless, the loss of generality brought about by the specialization of the model also implies that models are now potentially tight to a camera's context. Where we previously had a single model for the analysis of many cameras, we now potentially need one model per camera, each requiring its very own training process with properly annotated training examples. One training per camera quickly becomes unrealistic as the number of cameras grows, limiting the potential benefits of model specialization unless the process can be optimized and completely automated.

Neural networks aim to capture robust features from objects that are invariant across environments. This allows the network to correctly identify unseen objects even if they are captured in previously unseen contexts. For example, a flying pig is a pig even if the unexpected happens and it is seen flying. However, the hard truth is that the learning process of such robust features is reserved to those that are (1) able to gather massive training datasets and (2) willing to pay the computational cost of neural networks big enough to capture that level of generality, during both training and inference time. On the other hand, once we accept that even humans will have a hard time identifying a flying pig captured by a camera feed, we can also accept that the context matters. Context is a strong bias, indeed, but a helpful bias nonetheless that largely simplifies the problem. This simplification is especially important when working with the so-called edge neural networks; neural networks small enough to run in edge devices but also limited in the amount of generality they can retain. Hence, if we wanted to deploy a camera trap on a farm to track pigs and birds, we can safely make the trained model assume that if it flies, chances are it is not a flying pig. While it may seem as an extreme example, this can be extrapolated to daily scenarios where video analytics is commonly deployed. A camera in a racetrack may capture a *Formula 1* car, while an urban intersection does not need to know how such a vehicle looks. We call this *overfitting the context*. However, there is still the risk of the model defaulting predictions based *solely* on the context. To avoid the model defaulting to birds for any input image with the sky, we propose two techniques depending on the size and scope of the target model.

4.3 Problem Breakdown

Convolutional Neural Networks (CNNs) are able to learn abstract high-level representations of the objects they are trained to detect. The rule of thumb is that assuming proper training, the bigger the network, the better it can learn high-level features to generalize with high accuracy on new data. However, the deployment of generalist CNNs with state-of-the-art accuracy raises three main difficulties:

- They require a vast amount of data adequately labeled.
- The training phase often takes hundreds, if not thousands, of hours, even with high-end accelerators (such as GPUs or TPUs).
- Complex models may provide good accuracy but are not suitable to be executed on resource-constrained edge nodes, while simpler models suitable for the edge achieve lower out-of-the-box accuracy that is often not enough for many use cases.

Thanks to automating the retrieval, selection and annotation of the training images (even if it results in an imperfect annotation) and tailoring or specializing lightweight models, we can provide high accuracy models with real-time performance that can be reliably trained within minutes (or less than an hour on CPU-only).

The end-to-end process of specializing models is composed of various steps, each with its own set of challenges. Therefore, the automation of the process with satisfactory results requires several considerations to be made. This section breaks the problem of model specialization down into its core parts to analyze its challenges and considerations. Figure 4.2 depicts the different steps involved in the process. Once images are captured by the camera, the most representative are selected to be annotated by the ground-truth model and be included in the training dataset. Following, the edge model is trained using the generated dataset. Once deployed, the model's accuracy can be monitored to trigger new training iterations if it falls below a certain threshold.

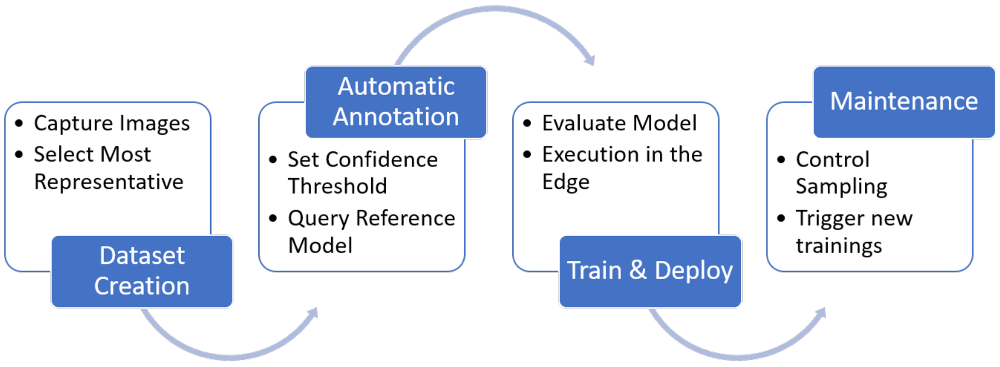


Figure 4.2: Breakdown of the different steps involved in the automation of specializing models.

4.3.1 Static Context from Static Cameras

Before we delve into the specifics of the problem and the solution we propose, we would like to highlight the main observation that makes everything possible: static cameras do not move. While this observation may seem obvious, it simplifies the problem and the solution to a great extent.

The training of object detection models requires a large number of diverse training examples. Object instances in the training dataset should appear in all their diversity (e.g., different colors, shapes, or perspectives). This diversity allows the neural network to learn high-level features inherent to the object class and invariant regardless of the specifics of the observed instance. In the end, diversity during training lets the model generalize. At the same time, a static camera is a camera whose position and orientation remain invariable throughout time. That is, it always points at the same scene using the same lens and from the same distance and point of view. Therefore, thanks to assuming static cameras, we are able to define the context of a camera and, then, tailor models for it.

A static camera thus allows the effective optimization of the pipeline by applying simple traditional computer vision techniques. Specifically, we make use of simple techniques for background subtraction with two purposes: (1) select for training and annotation only those frames where there was any meaningful change and (2) let the *ground-truth* model use the image resolution to its full extent by focusing on the parts that changed.

4.3.2 Capture (Relevant) Training Images

Each image annotated by the ground-truth model has a cost associated in terms of network bandwidth and compute power in the data center. However, not all images captured by a camera provide the same amount of new information (if any, at all). Video streams from static cameras tend to have a high degree of *temporal locality* as frame rates between 10 to 30 frames per second are standard for edge cameras. Such frame rates translate to new images being captured every 100ms to as low as 30ms. In most scenarios, nothing new happens within 30 milliseconds, but simply increasing the time between frames does not guarantee that new objects enter the scene. Therefore, we should find the right set of images that will maximize the accuracy of the resultant edge model while minimizing the number of queries to the data center.

Luckily, we can exploit the temporal locality to easily detect changes or quantify similarity between consecutive frames. As the scene's background remains mostly still throughout time, change can be detected by simply comparing frames and set a threshold to decide whether a frame should be considered or not. Relatively simple and inexpensive distance metrics, such as the Mean Squared Error (MSE), can be used to detect meaningful changes between frames [55], while other mechanisms, more robust but also more computationally expensive, rely on Siamese neural networks [25].

Furthermore, thanks to the same temporal locality, we can also leverage robust motion detection and background subtraction techniques. When working with static cameras, we can consider change to be caused by movement. Therefore, we can identify and isolate the regions of the scene that contain movement, i.e., regions of interest. For such purposes, different approaches with different accuracy-cost trade-offs are available, from those that make use of traditional computer vision techniques [15] to more modern that make use of neural networks [18].

COVA, as detailed in Section 4.4.1, makes use of traditional computer vision techniques to model the background and detect and extract the regions containing movement. These regions are regularly smaller than the full scene and can be compared against the same region in previous frames to quantify the level of similarity at a finer grain. Moreover, the extraction of such regions brings an unexpected benefit that, according to our results, boosts the accuracy of the annotations [78]. Nowadays, FullHD video (1920x1080 pixels), if not 4K (3840x2160 pixels), is widely used even among inexpensive cameras available on the market. However, neural networks are rarely deployed with an input resolution matching the camera's resolution, as the computational cost of their execution would become intractable. However, it does not appear to be a limitation on technology but rather a matter of diminishing returns. On the contrary, neural networks do not require large pixel densities to distinguish objects of different classes. An input size of 768x768x3 (as the ground-truth model used throughout the chapter) is above average, judging by what can be found in the literature [93], while 300x300x3 pixels is a typical size for edge models (like the one used throughout the chapter).

If we consider object detection in the outside world, small input sizes may indeed become a problem. In these scenarios, objects are captured from arbitrarily large distances, and the smaller objects quickly turn into a small set of indistinguishable pixels as the image is resized to the neural network's input size. In such cases, it is worth considering the image to its full extent. The same motion detection techniques we use to optimize the collection of images allow us to do this. Once motion is detected, COVA crops the bounding box containing all frame regions that changed with respect to the background model. Consequently, regions that did not add any information are discarded, and the model can focus on detecting whatever was that moved. For example, before passing the image to the ground-truth model, we have compressed the information captured by the camera between 3.5 and 14 times for FullHD and 4K images, respectively (or between 23 and 92 times for the edge model). As Figure 4.3 shows, the average percentage of the frame's area containing motion in different scenarios is often a small fraction of the total area. According to the results, these regions represent as little as 3% of the scene's area.

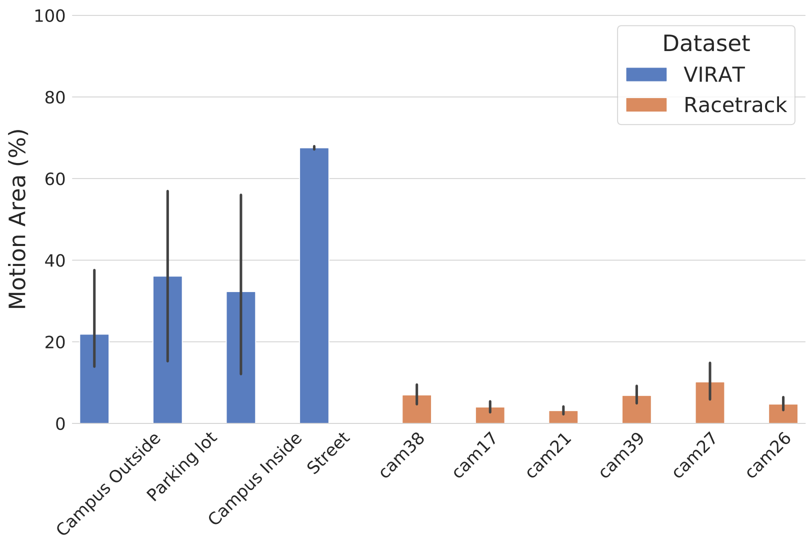


Figure 4.3: Average percentage of the frame's area containing motion for different scenes. Frames from the Racetrack dataset have much smaller areas, as cars in the track can only drive through the asphalt and always follow the same paths. The VIRAT dataset contains scenes with up to 67% of the area covered by objects of interest, common in busy urban scenarios with objects appearing from multiple points.

4.3.3 Model Specialization

To deploy video analytics to multiple cameras (each one with its own context), we could take two main directions. On the one hand, we could run big and complex CNNs capable of generalizing and delivering high accuracy regardless of the cameras' context. This solution could work out-of-the-box if the model has already been trained to detect the same set (or a superset) of object classes as those we want to detect in the edge. Moreover, this solution would require a single training for multiple cameras. However, the reality is that edge deployments do not have the luxury to run such models whose compute requirements are rarely met at the edge. Accuracy comes at the price of high latency and low throughput, often below the minimum required to run real-time use cases. On the other hand, we can use lightweight neural networks but *specialize* them to each specific deployment and let them deliver their full potential on a narrower scope.

The edge cloud is limited on the size of the neural networks that can run. At the same time, this puts a limit on the level of accuracy that a model can achieve with respect to that in the state-of-the-art [16]. Hence, models deployed to the edge may face difficulties extrapolating their knowledge to detect objects in new environments. The specific lighting conditions, points of view, or focal distance of every deployment may impact the ability of small neural networks to extrapolate using previously learned features. One approach to mitigate the effects of lack of generalization from smaller models is called *model specialization*, which implies narrowing the scope of what the network has to learn and, thus, demanding less from it. In the context of edge video analytics, model specialization allows us to adapt a model for the specific context of each camera deployment. The specifics of a context may not be known beforehand but can be easily extracted once the camera has been deployed and begins to capture images.

Model specialization can be achieved through different means. One option is to train the network from scratch on a specialized dataset. However, the computational cost plus the number of labeled training images required to train a network from scratch on a per-camera basis makes it unfeasible for rapid deployments. Nevertheless, different techniques, such as model distillation [42] or transfer learning, allow us to use previously learned knowledge and repurpose it faster while requiring fewer training images. In this chapter, we focus on transfer learning to specialize pre-trained models on the specifics of each deployment. The knowledge of a pre-trained generic model with state-of-the-art accuracy, working as *teacher* or *ground-truth model*, is distilled into a specialized training dataset used to train the *student* or *edge model*.

Nonetheless, a neural network's predictions are as good as the dataset used to train the network, and transfer learning still requires properly labeled images to specialize models. A proper training dataset is essential to produce an accurate yet small model that can process images in real-time on constrained nodes.

4.3.4 Concept Drift Detection

Concept drift refers to a change in the statistical properties of the variable that a model tries to predict. In the context of edge deployments, concept drift could arise in different ways. For example, a camera in the wild would capture predominantly green backgrounds during spring that turn white during winter due to snow; a camera whose perspective changes substantially after it is relocated; a change in light conditions from day to night; or simply because the user is interested in new classes. A new training should be triggered to adapt the models to the new environment or contextual features in all these cases.

This said, concept drift detection is not trivial. Traditional approaches try to detect concept drift by performing statistical hypothesis testing on multivariate data [28], while more recent ones explicitly designed for neural networks propose the use of an oracle or teacher model that is consulted intermittently to asses the deployed model's accuracy [63] and drive the training rate whenever accuracy falls below a threshold.

Nonetheless, it is important to highlight how, in the context of static cameras, concept drift is highly coupled to the level of specialization of the deployed model. A highly-specialized model trained using only daytime examples may face difficulties in detecting the same objects during nighttime due to lighting variations. On the contrary, a model trained with all-day examples and big enough to retain the sufficient level of generality will not experience a concept drift as day turns to night. Consequently, prediction cascades can help mitigate the problems associated with concept drift without requiring retraining. In prediction cascades, multiple specialized models (either binary [55] or multi-label classifiers [102]) are sorted by decreasing level of specialization (or increasing level of accuracy and cost) and a model is only consulted if the previous one is not confident enough of its predictions. In the event of a short-term concept drift that cannot be absorbed by the more specialized models, prediction cascades use the more expensive models as fallback to ensure the accuracy of the predictions. However, in the event of a long-term concept drift, the specialized models will be rendered useless and counterproductive, as the cascade will keep consulting them even when they can no longer produce reliable predictions. Therefore, prediction cascades can mitigate the problem but do not solve it. As such, they are a better suited for query systems where the model search can consider the full video stream being queried.

4.3.5 Incremental Learning

The detection of a concept drift irredeemably results in a new training of the model to adapt it to the new environment. This is especially true in ever-changing scenarios, like seasons in a forest, where we have to rely on *incremental learning* to train a model with whatever data is available at the time of the deployment and, as new examples appear, retrain the model using the newly captured examples. However, there is one crucial limitation: *catastrophic forgetting*[36], which refers to the tendency of neural networks to forget previously learned features entirely upon learning new ones. The straightforward solution is to train again using the entire dataset, i.e., include the newly captured examples into to the training dataset used during previous iterations. However, this can lead to an oversized dataset that can quickly become intractable for edge nodes, while older examples may not be significant anymore in the current environment. When the set of classes that are missing is known, another solution is to add generic instances for those classes (e.g., using online image search engines [58] or reusing a subset of the examples used during the training of the ground-truth model or other *students*). Nonetheless, previous work demonstrates that using knowledge distilling techniques [42] networks can retain most of the previously learned features, even if trained only on new classes of objects [57, 85].

4.3.6 Considerations on Automatic Annotation

The bigger models (potentially) have higher accuracy because they have more trainable parameters. When properly trained, the extra parameters can learn more and better features. These may come in the form of more layers, but the easiest way to increase accuracy (at the expense of computational cost) is to increase the input size [44]. A larger input size increases the amount of information that the network receives in the form of pixels and allows models to *see* objects bigger, meaning the network's receptive field takes more pixels into account which is particularly important when trying to detect small objects in the background.

Experiments show that mispredictions usually come from the model's inability to distinguish the object from the background (i.e., while Top-1 or Top-5 recall might be good, the confidence is below the threshold to be considered) [43]. In other words, the model may correctly distinguish an object from other objects but not be sure whether the object is actually there. This phenomenon increases as the object gets smaller and becomes just a *few pixels* large. Figure 4.4 shows the impact on the confidence of predictions with respect to the size of the input image using different pre-trained models. The input images belong to different frames captured from one scene of the *Racetrack* dataset, all containing a single car centered in the scene. For each frame, the region of interest (RoI) containing the car object is cropped in different sizes and fed to the model. Therefore, the car represents a smaller area of the model's input as we increase the RoI size. The experiment evaluates models with three different neural network architectures: SSD detector with MobileNet V2 feature extractor, SSD detector with ResNet101 feature extractor, and EfficientDet (also SSD with EfficientNet feature extractor). The models are different pre-trained versions of these three neural network architectures that were trained and dimensioned with different input sizes. These results show that confidence drops as the object gets smaller with respect to the rest of the scene. There are two factors contributing to this effect. First, the model's input size. Intuitively, bigger input sizes have a larger receptive field and are, potentially, better at locating small objects. Second, the neural network architecture. While the more modest MobileNet achieves on-par confidence compared to the state-of-the-art EfficientDet when the car represents the largest part of the input image, its confidence quickly drops as the object becomes smaller regardless of its input size. On the contrary, the ResNet101 and the EfficientNet backbones seem to sustain their confidence as the object gets smaller. However, their confidences still drop for the larger RoI sizes, especially for the models with smaller input sizes.

Finally, the *ground-truth* model is assumed to be knowledgeable of the scope in which

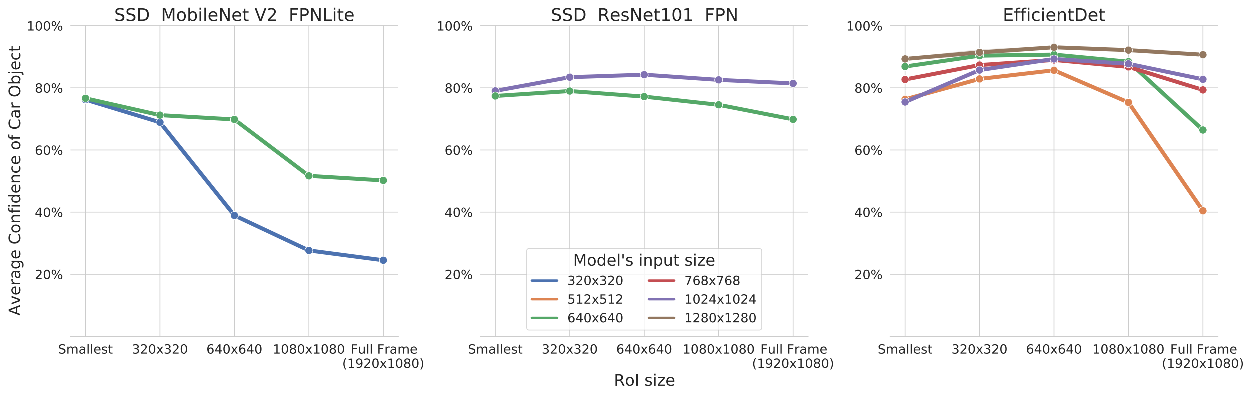


Figure 4.4: Average confidence of car detections from three models with different input sizes and varying the size of the Region of Interest (RoI) passed as inputs. All RoIs contain a car centered. *Smallest* refers to the smallest square RoI that contains the car object, i.e., the smallest bounding box with aspect ratio of 1 that contains the full car object. *Full Frame* refers to the full frame passed to the model as input at full resolution (1920x1080), i.e., as it was captured by the camera.

edge deployments will happen but is not required to be trained on images from those deployments. We build this assumption upon the observation that large models are better at retaining features that allow them to generalize better. These models are therefore expected to correctly identify *most* of the objects from classes they have been trained to detect, even if these objects are seen from new perspectives and distances. This level of generalization is a characteristic that models small enough to run on edge devices do not share.

4.4 The COVA Framework

COVA is a framework that automates the task of specializing neural network models for edge video analytics. As a framework, it aims to enable the rapid deployment of customized applications in resource-constrained edge nodes. As such, it has been implemented with customization and extensibility in mind.

COVA is made out of two main building blocks. On one side, it provides high-level structures to allow video analytics models to be tailored for a specific deployment in just a few lines of code. On the other side, it also provides multiple built-in methods to ease the extension and customization of the existing structures.

4.4.1 Framework Architecture

The backbone component in the workflow of COVA is the pipeline (or *COVAPipeline*). A pipeline describes the sequence of steps involved in specializing a neural network model and how each step communicates with the next. Together, a sequence of steps conforms a pipeline. Figure 4.5 depicts the default pipeline implemented in COVA entailing five steps. These are 1. capture images, 2. filter images, 3. annotate images, 4. create the dataset, and 5. training. The default pipeline iterates over steps 1-3 until enough images are available to continue. Then, step 4 generates the training dataset from the images and the annotations from the previous steps. Finally, the model is trained in step 5.

The potential of COVA relies on the high level of customization allowed by the pipeline architecture while keeping a simple structure within its core. Each step in the pipeline is implemented using a plugin architecture, allowing to easily extend or modify the default behavior. For example, the annotation step can be carried out using the built-in server exposing a REST API or the annotation plugin offloading the task to Amazon Web Services. Moreover, any step implementation (i.e., plugin) is interchangeable with any other implementation of the same step as long as their interfaces match what is described in the pipeline definition. Furthermore, it is possible to define different pipelines that use different stages and plugins and do so only through the configuration file, i.e., without any modification in the code.

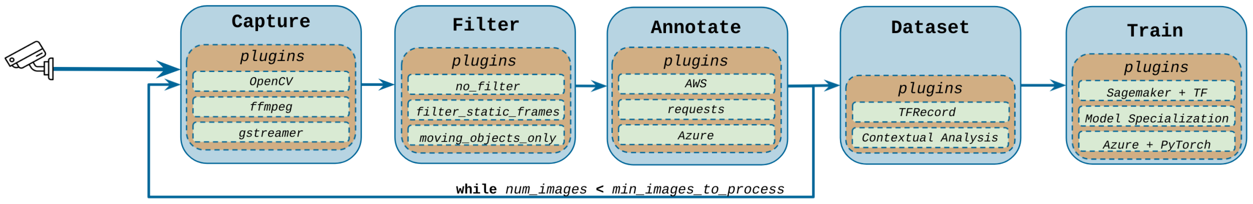


Figure 4.5: Default pipeline in COVA for automatic specialization of models.

4.4.2 Pipeline Components

By default, COVA implements a pipeline with the five stages previously described. However, the pipeline itself can be extended or modified if another sequence of stages is needed. Nonetheless, we describe and focus on the default pipeline, its stages, and the built-in plugins.

COVACapture. Captures and decodes images from a stream (input) and returns decoded frames as an RGB matrix. COVA implements capture using OpenCV, which, in turn, accepts different backends such as FFmpeg or GStreamer.

COVAFilter. Filters decoded frames (input) and returns a list of RGB matrices. Among the built-in filters, *no_filter* returns the same frame it received as input, i.e., does not apply any filtering; *filter_static_frames* returns either the same input frame or an empty list, depending on whether any movement was detected in the latest frame or not; *moving_objects_only* returns a list of the regions of the input frame where movement was detected.

COVAAnnotate. Annotates images. It receives filtered frames and returns the list of bounding boxes and labels of the objects detected on the input images. COVA implements two annotation methods: using the built-in REST API or using AWS. Both methods assume a ground-truth model trained on the target problem whose predictions are assumed to be ground-truths during training. Section 4.6 evaluates the impact of this assumption. This stage supports both TensorFlow and OpenVINO models.

COVADataset. Generates the training dataset from the filtered images and their respective annotations. The built-in plugin results in a TFRecord file as output (format used in TensorFlow that stores data as a sequence of binary strings). However, the resulting dataset can be subject to different processes. Specifically, COVA implements a default dataset in which all annotations make it into the dataset (since filtering occurs in the Filter stage).

COVATrain. Trains a neural network using the previously generated training dataset. TensorFlow is the default library used. Built-in plugins support two flavors of training: standard TensorFlow and TensorFlow Object Detection API (used for training or fine-tuning object detection models). Both flavors can be executed either locally or using AWS. As output, this stage generates a specialized model (in TensorFlow's *SavedModel* format).

4.4.3 Pipeline Configuration

COVA allows pipelines to be fully defined using a configuration file in JSON format. The configuration file allows users to dynamically define the plugins to be used on each pipeline stage and the parameters expected by each stage. Figure 4.6 shows an example of a pipeline defined using a configuration file in JSON format. Custom plugins can be loaded in the configuration file using the *plugin_path* keyword. The set of parameters accepted by each plugin will ultimately depend on the specific plugin implementation. However, some of the most important ones include:

```
"capture": {
    "plugin": "VideoCapture",
    "args": {
        "stream": "rtmp://example.com/stream",
        "frameskip": 25,
        "deadline_max_images": 1000,
        "deadline_max_seconds": 3600
    }
},
"filter": {
    "plugin": "FilterStatic",
    "args": { "warmup_seconds": 10 }
},
"annotate": {
    "plugin": "AWSAnnotation",
    "args": {
        "aws_config": {
            "role": "$globals#aws_role",
            "instance_type": "ml.m4.xlarge"
        }
    }
},
"dataset": {
    "plugin": "AWSDataset",
    "args": {
        "valid_classes": [
            "car", "person", "motorcycle",
            "bicycle", "truck", "bus"],
        "min_score": 0.3
    }
},
"train": {
    "plugin": "TFTrainer",
    "args": {
        "checkpoint": "models/ssd-mobilenetv2/",
        "epochs": 1000,
        "layers_to_train": "box_regression"
    }
}
```

Figure 4.6: Example of configuration file defining a COVA pipeline.

Target classes of objects. List of object classes for which the model will be specialized. The list of classes is the basis of model specialization for two reasons. On the one hand, the type of objects expected to be seen and detected in the scene is essential to define a camera's context. Limiting the classes of objects the model has to learn to identify helps specialize the resulting model, and the learning process can focus on those classes it will eventually encounter. On the other hand, the precision of the ground-truth model gets a boost after we discard detections from classes that are unlikely to be seen in a specific context (e.g., a boat detected crossing a pedestrian crossing can be omitted or, at the very least, flagged to be double-checked). Only frames containing instances of one of the target classes are considered during the dataset's creation, while the rest are filtered out.

Training deadline. It is important to set a deadline to force the trigger of the training process in scenarios where there is no high influx of new objects of interest entering the scene. If the training is triggered by the deadline, results may not be optimal, but, at least, the model can start working while new images are being captured for a new training iteration.

Motion sensitiveness. If the filter plugin uses motion detection, it can be tweaked to be more or less sensitive to small changes in the frame. This parameter will highly depend on the characteristics of the scene. The smaller the objects we want to capture, the higher the sensitivity required to detect their movement. However, high sensitivity may incur an undesirably large number of frames being flagged with motion, as small variations can occur due to glitches, camera jitter, or the wind blowing on background objects.

Trainable Layers. If the training plugin uses a checkpoint of a pre-trained model, COVA allows defining the layers that are to be trained while the rest are frozen. We have tested three configurations: *unfrozen* (all weights are trainable), box regression (only the layers of the object detector head are trainable), and top (box regression plus the last few layers of the feature extractor). According to our experiments, the optimal configuration will highly depend on the size of the training dataset. With enough examples, unfrozen seems to yield the best results. However, box regression is the safest option and yields good results even with only tens of images, while unfrozen has a higher risk of overfitting with small datasets.

4.4.4 Built-in Utilities

COVA also provides several built-in methods aimed at assisting with the development of new applications. Among all the utilities and auxiliary methods, we primarily focus on motion detection, as it is the basis of how COVA analyzes the scene to provide better results while reducing the amount of network bandwidth (as only a portion of the frame needs to be sent) and compute used (by reducing the amount of annotations required).

Motion Detection

There is a myriad of existing methods to perform motion detection. Nonetheless, it involves many challenges (e.g., camera jitter or lighting variations), and, as a problem, it is far from being solved. There is not a *de-facto* approach that can handle all scenarios robustly, and different methods offer different trade-offs in terms of speed or accuracy. In COVA, we prioritized speed over accuracy when selecting the methods to implement for two reasons. First, the accuracy of these methods is usually evaluated using image quality metrics of the generated background or error metrics at the pixel level to compare the generated background to the ground-truth background. Second, COVA must run on resource-constrained Edge nodes and do it faster and more cost-effectively than the alternative: send whole frames over the network and process them in a centralized data center. Therefore, COVA implements two simple methods consisting of two steps or building blocks:

Background Modeling of the scene is the key step for motion detection to work effectively. Take an incorrect background of the scene, and any advantage from motion detection fades away, as every single frame could be incorrectly flagged as containing changes. As previously mentioned, COVA assumes images from static cameras, which greatly simplifies the background modeling. Nonetheless, the optimal algorithm will still be highly dependant on the specific context in which the camera is placed. Therefore, COVA implements several methods. The first and simplest is to assume that the first frame captured is the background of the scene. This method works best when the scene is known to be empty of moving objects when the process starts, and the background is not subject to long-term changes. However, in many cases, this is not only false, but the definition of background can change over time (e.g., sunlight turning into darkness or objects left unattended for long periods, like parked cars). Therefore, background modeling works best when the background is computed over time. In this direction, COVA implements background modeling using a Mixture of Gaussians (MoGs) [88]. Our results have shown that this method yields robust results while being computationally inexpensive (i.e., on average, 2.7 milliseconds on a single core of an Intel Xeon 4114). This is the method used by default.

Monitor the scene for any substantial changes. A simple subtraction operation will give us the pixel-to-pixel absolute difference between the current frame and the background model. Nonetheless, in real-world deployments, two consecutive frames will never be identical. Multiple factors will cause small pixel changes even with a static background and no objects moving.

On the one hand, digital camera sensors are imperfect and introduce tiny variations that cause pixel values to vary, even when the scene remains seemingly invariant. Applying a Gaussian blur to both the background and the current frame filters noise out when computing the pixel-to-pixel difference. Next, COVA applies a binary threshold to the computed difference to consider only those regions containing significant changes in the pixel intensity. After applying a *dilate* operation to fill in the holes over the result of the binary threshold, we obtain a mask with the changing regions in the frame. Finally, we compute the bounding boxes by finding the contours on the masked frame. A contour is a curve joining all the adjacent points along the boundary of a region of pixels with the same color or intensity.

On the other hand, small changes can still be undesirably (although not necessarily incorrectly) flagged as motion. There is no rule of thumb to determine how large a region must be to be considered a potential moving object, as it will highly depend on the specifics of the scene and the objects we want to detect. For example, in the *Racetrack* dataset, some cameras point directly to the track at a relatively short distance, and cars are seen big enough to consider only large regions as motion. However, other cameras have a broader view and capture cars and other objects from different distances, and, area-wise, a distant car might be indistinguishable from a leaf fluttered by the breeze from a nearby tree. Therefore, COVA considers a user-defined threshold that sets the minimum area for regions to be considered of interest.

Figure 4.7 depicts an example of the process, where two cars are correctly detected as the only two objects moving in the scene. Each one of these operations is already implemented by OpenCV [66]. Altogether, they result in an inexpensive yet effective method for our purpose. Through them, motion detection takes orders of magnitude less time than computing a single inference on resource-constrained nodes. Therefore, it becomes ideal to be executed on resource-constrained nodes on a frame-by-frame basis to optimize the automatic annotation of images, which is indeed the most expensive single operation in the whole process.

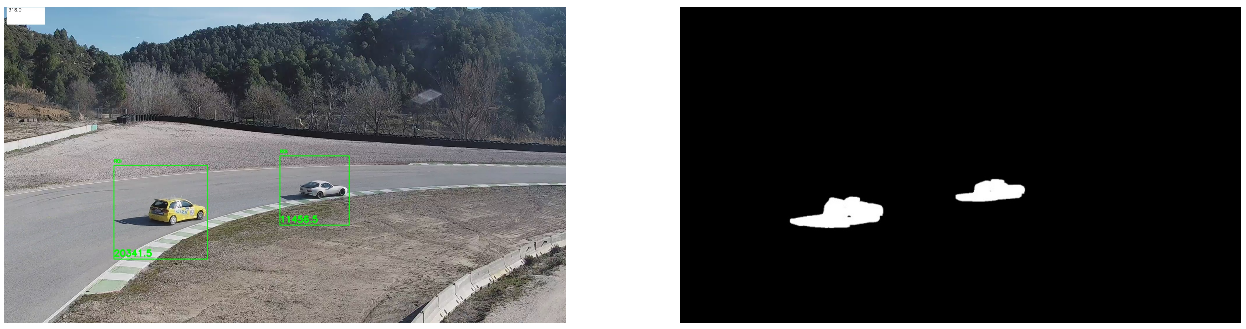
Cloud-Assisted Edge Deployments

As part of the Edge-Cloud interplay that COVA aims to leverage, some of the built-in plugins already implement the usage of Amazon Web Services. Specifically, S3 to upload images and *SageMaker* for the annotation and training stages. Moreover, COVA provides several auxiliary modules and utilities to integrate these services on new applications.

4.4.5 Source Code Availability

The entirety of the COVA project has been made publicly available in the project's repository [77] with an open license (Apache License 2.0) to encourage the community to use it, test it, and extend it. The repository includes the fully documented source code with detailed instructions on deploying it end-to-end and reproducing the experiments described in this chapter.

COVA is implemented in Python. By default, its only requirements are OpenCV and TensorFlow, while CUDA can be used if available.



(a) (b)

Figure 4.7: (a) Two cars in the racetrack with bounding boxes around them, obtained after performing background subtraction on the frame. (b) Thanks to a static background, the task of separating moving objects from the background is largely simplified. The regions of interest after applying simple computer vision techniques are much smaller than the whole FullHD frame captured by the camera lens, which allows us to focus the attention closer to the objects and increase the accuracy of the ground-truth model. In a scenario like the racetrack of the image, motion detection alone is able to filter all empty frames out and only consider those with cars and minimize the annotation costs.

Model	**Variant**	**Trained/Fine-Tuned on**
ground-truth	off-the-shelf	pre-trained on MS COCO
	generic	other similar edge scenarios
Edge	off-the-shelf	pre-trained on MS COCO
	generic	other similar edge scenarios
	specialized	the same camera of deployment

Table 4.2: Summary table of the model taxonomy with the different types of models considered throughout the chapter.

4.5 Methodology

In this section, we detail the methodology followed throughout the experiments to evaluate their results.

4.5.1 Edge Model Taxonomy

The evaluation in Section 4.6 is carried out using two neural network architectures and three different versions of each one. Table 4.2 describes the model taxonomy used throughout the remainder of the chapter. The *edge model* refers to the model that we aim to optimize (i.e., specialize) for its deployment to the edge. It is typically a lightweight model that can run at real-time performance on resource-constrained nodes. The *ground-truth model* refers to the model whose predictions are considered to be ground-truths while building the dataset to use during the training phase of the edge model. Edge and ground-truth are conceptual classifications for two models that play different roles within our proposed solution. Therefore, these do not refer to any specific neural network architectures, and any model can be used as either edge or ground-truth models. However, the ground-truth model is expected to be more accurate and more complex than the edge model. Otherwise, the latter would be a better choice to deploy to the edge.

Moreover, we define three different variants that refer to how the model was trained: *off-*

the-shelf, generic, and specialized. Off-the-shelf (OTS) refers to the model pre-trained on the COCO dataset (Common Objects in Context) [60] without any further training nor tuning on any other context. It is easy to find most of the well-known neural network architectures already trained on this dataset. Generic and specialist variants are categories always relative to the model to deploy. Generic refers to a version of the model trained or fine-tuned on contexts similar to the deployment context. For example, if we want to deploy a model to control cars at a parking lot, a generic model could have been trained using images from other parking lots or urban areas with mostly cars. Specialized refers to a model that has been trained using images from the same edge camera of the deployment.

4.5.2 Datasets

Throughout this chapter, we use two datasets to evaluate the different points of our approach.

- VIRAT Video Dataset [65]. Data collected in "*in natural scenes showing people performing normal actions in standard contexts, with uncontrolled, cluttered backgrounds*". After filtering scenes that were not fully annotated, the dataset we used contains a total of 10 scenes from 10 different static cameras. We have grouped these 10 scenes into 4 contexts: *Campus Inside* for scenes 0000, 0001, and 0002; *Campus Outside* for scenes 0100, 0101, and 0102; *Parking Lot* for scenes 0401 and 0403; and *Street* for scenes 0501 and 0503. The name of these groups is descriptive of the context in which the images were captured.
- Racetrack. Custom dataset with footage captured during car training session in a racetrack. The dataset includes images from 6 static cameras placed in different spots along the track and captured at 12 different points in time. This dataset is not fully annotated, and we use it only for a meta-analysis of its characteristics. It is nonetheless interesting, as it gives the perspective of a use case in which cars are virtually the only moving objects crossing the field of vision of the cameras.

In the experiments presented in Section 4.6, we focus the evaluation on the VIRAT dataset, as we consider it to present enough diversity of scenes to be representative.

4.5.3 Evaluation Metrics

In image classification, the two main metrics to evaluate a model are *precision* and *recall*. For a certain class of objects, precision will tell us how much we can trust that a given prediction has been correctly classified (i.e., the ratio of True Positives with respect to the total of predictions), while recall will tell us how much we can trust that images of a certain class are correctly classified (i.e., the ratio of True Positives with respect to the total number of *ground-truth* True Positives). However, the goal of object detection models is to find the bounding box coordinates of each detected object, and these coordinates are real numbers. Therefore, to classify a prediction as True or False Positive and Negatives, we consider the Intersection over Union (IoU) between the predicted box and the ground-truth. The detection is considered a True Positive if the computed IoU is over a certain threshold and is considered False Positive otherwise.

Mean Average Precision (mAP)

We use mAP (mean Average Precision) as the main metric to evaluate each version of the different models resulting from the experiments. Within object detection, mAP has gained popularity as it is used in some of the most popular challenges in the field, such as PASCAL Visual Object Classes (VOC) [32] and MS Common Objects in Context (COCO) [60]. However, the details of how mAP is computed may vary from challenge to challenge. In this paper, we compute mAP as specified by the COCO challenge.

4.5.4 Experiment Setup

The experiments explore the different considerations we have identified. However, there are a series of parameters that are common throughout the different experiments.

- Transfer learning affects only the parameters on the box regression layers of the model, i.e., box regression layers are left unfrozen, while feature extraction layers are frozen.
- Size of the dataset: collection stops at 1000 training images *per scene*. The evaluation dataset contains two hundred images per scene. Training and evaluation images are extracted from different videos, i.e., they contain images from different points in time to reduce overlapping.
- Models are trained for 5000 epochs.

Baseline OTS Models

For the evaluation, we start from an edge and ground-truth *off-the-shelf* models. These models are publicly accessible from the TF Model Zoo [93] and have been pre-trained on the COCO (Common Objects in COntext) dataset [60]. For the experiments, we have selected a MobileNetV2 feature extractor with a Single Shot Detector (SSD) detection head [91] with an input size of 300x300x3 pixels as the *edge* model and an EfficientNet feature extractor with an SSD head [90] and an input size of 768x768x3 pixels as our *ground-truth* model. The edge model was selected for being the smallest and fastest pre-trained model available with a while the ground-truth model was selected for being on the high-end of best performing models. The mAP of the models on the COCO dataset is 20.2% and 41.8%, for the edge and ground-truth models, respectively.

4.6 Results

We have broken down the process of automatic tuning of neural networks into different steps. Each step invariably brings a trade-off that we must understand before moving forward.

The following experiments are designed to test the hypotheses on which COVA is based. First, we test the level of generalization *off-the-shelf* achieved by the ground-truth model compared with the edge model. Then, we analyze the potential impact of specialization on accuracy by comparing the accuracy of generalist models with that of models specialized for a specific scene or context. Finally, we evaluate the impact that an automatic annotation of the training dataset has on the accuracy of the resulting specialized models, i.e., the accuracy drop with respect to manual annotation, which is assumed to be perfect.

4.6.1 Off-the-Shelf Generalization

This experiment tests the hypothesis that larger models are able to generalize better than smaller models, i.e., achieve higher accuracy without being fine-tuned or trained on the specific context on which we evaluate them.

Model	Category	COCO mAP	VIRAT mAP	Diff
EfficientDetD3	Ground-truth	45.4	46.7	+3%
EfficientDetD2	Ground-truth	41.8	41.7	-0.25%
R-CNN ResNet152	Ground-truth	39.6	43.5	+10%
SSD MobilenetV2 FPNLite	Edge	22.2	20.5	-8%
SSD MobilenetV2	Edge	20.2	15.4	-25%

Table 4.3: Comparison of mean Average Precision on the COCO and VIRAT (average of all scenes) datasets for different *off-the-shelf* ground-truth and edge models.

Table 4.3 shows the average mAP achieved by different ground-truth and edge models *off-the-shelf*, i.e., pre-trained on the COCO dataset. All three ground-truth models evaluated achieve similar or higher scores than what they achieved on the dataset on which they were trained. However, both edge models struggle to generalize to the VIRAT dataset using their pre-trained knowledge and drop their accuracy between 8% and 25%. Figure 4.8 zooms into the mAP of all models broken down by scene. Again, all ground-truth models consistently outperform the edge models on all scenes. The difference between ground-truth and edge models fluctuates between 2x and 18x, depending on the scene. On average, the ground-truth models outperform the edge models by a factor of 6x. These results highlight two things. First, the ground-truth models achieve significantly better accuracy than the smaller edge models, as expected. Second, ground-truth models seem to sustain their accuracy in new environments, implying a higher generalization level.

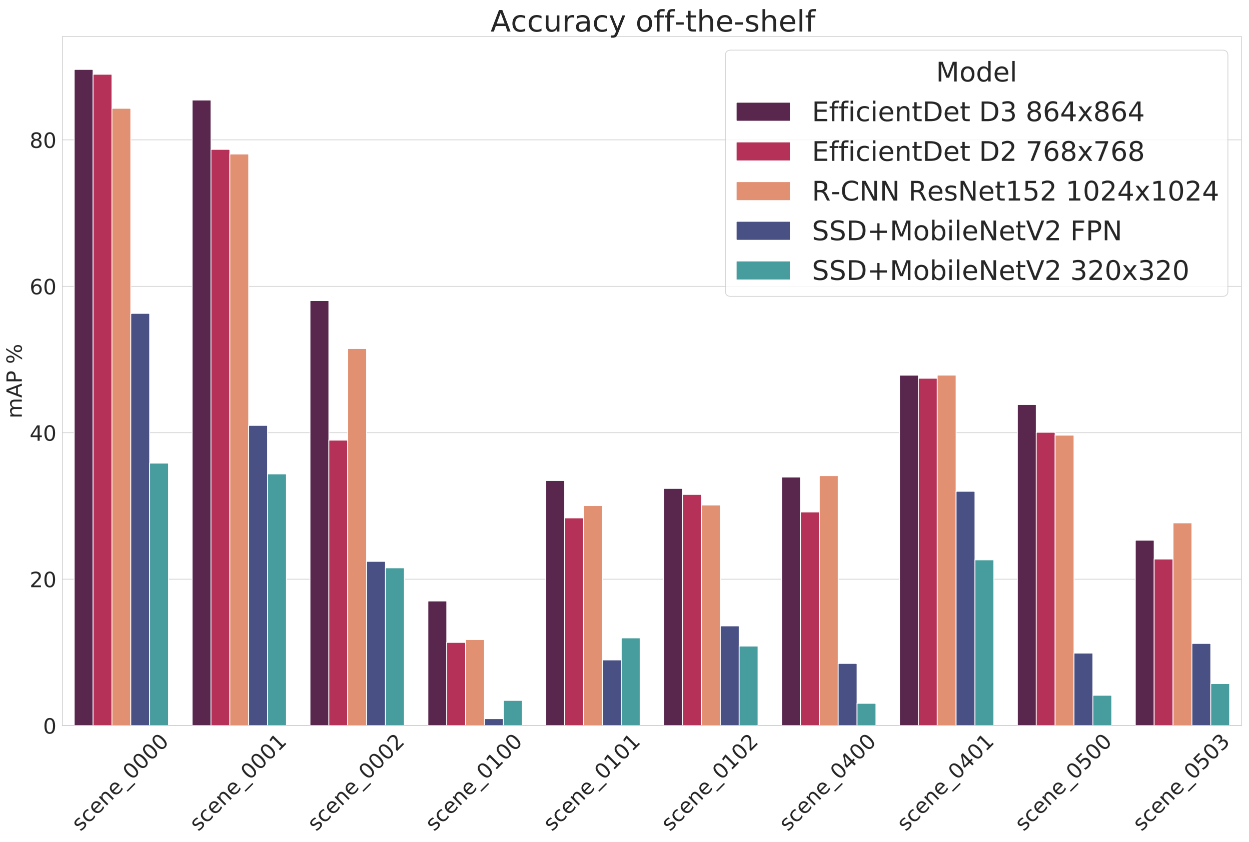


Figure 4.8: Mean Average Precision of three ground-truth models and two edge models *off-the-shelf* evaluated on the VIRAT dataset. Breakdown by scene.

4.6.2 Generic vs Specialized

This experiment tests the hypothesis that a specialized model performs better than a generic model when evaluated on the same context of its deployment.

We define the *generic* edge model as a version of the edge model that has been fine-tuned using images from scenes similar to the one we want to deploy the model. That is, images from other cameras different than the one to evaluate but belonging to the same supercategory as used in Section 4.3 (i.e., Campus Outside, Campus Inside, Parking Lot, Street). Analogously, we define the *specialized* edge model as a version of the edge model that has been fine-tuned using images from the context to deploy. That is, the model is fine-tuned using images from a single camera. Nonetheless, images on the training and evaluation datasets are from videos captured at different times, i.e., a video is used for either training or evaluation dataset, but never in both.

Figure 4.9 shows the mAP of the generic and the specialized edge models. On the one hand, the *off-the-shelf* models consistently underperform both generic and specialized models except on one scene. In scene 0102 the pre-trained model achieves 10% higher mAP than the generic and 15% higher than the specialized. Diving into the breakdown accuracy by object classes, the specialized achieves higher accuracy on person detection but still drops the average accuracy due to poor car detection. Upon closer inspection of the contents of the scene, we observed that the scene (supercategory *Campus Inside*) contains a single instance of a car, which was seemingly insufficient during training. On the other hand, the specialized model outperforms the generic model on every scene of the dataset (except, again, scene 0102) by a minimum of 2% and up to 87%, with an average improvement of 30%. These results prove how model specialization can help us achieve greater accuracy without altering the computational complexity of the model. However, they also highlight that special care should be taken in scenes where there is an imbalance on the class distribution of the objects captured.

4.6.3 Manual Ground-truth vs. Automatic Annotation

This experiment evaluates the impact of automatic annotation on the resulting model's accuracy. The edge model is trained using the same set of examples whose annotations were obtained using three annotation variants. On the one hand, manual annotation is taken as the reference, as it is assumed to be perfect. On the other hand, automatic annotation in which predictions from the ground-truth model are taken as ground-truths during training. Automatic annotation is evaluated with a minimum confidence threshold of 0.3 and 0.5. Therefore, predictions from the ground-truth model with lower confidence are discarded. Moreover, the ground-truth model is assumed to be trained on (or, at least, knowledgeable of) the classes of objects that will have to annotate and detect.

Figure 4.10 shows the mean Average Precision of the specialized edge model using three datasets annotated differently: manual, automatic (conf=0.3) and automatic (conf=0.5). Compared to manual annotation, the automatic annotation yields similar results on some scenes while it lags behind on others. Upon closer inspection of the different metrics, car objects' weight in the scene seems to be directly related to the quality (i.e., mAP) of the resulting models trained using automatic annotation. A closer look at the composition of the scenes where automatic annotation lacks significantly behind manual annotation hinted that the problem lies in the smaller objects that dominate those scenes, like person instances. Small objects are usually more difficult to detect for obvious reasons, and this translates to second-order errors, as the ground-truth model introduces annotation errors (mostly in the form of False Negatives by not detecting them). Consequently, the edge model fails to detect

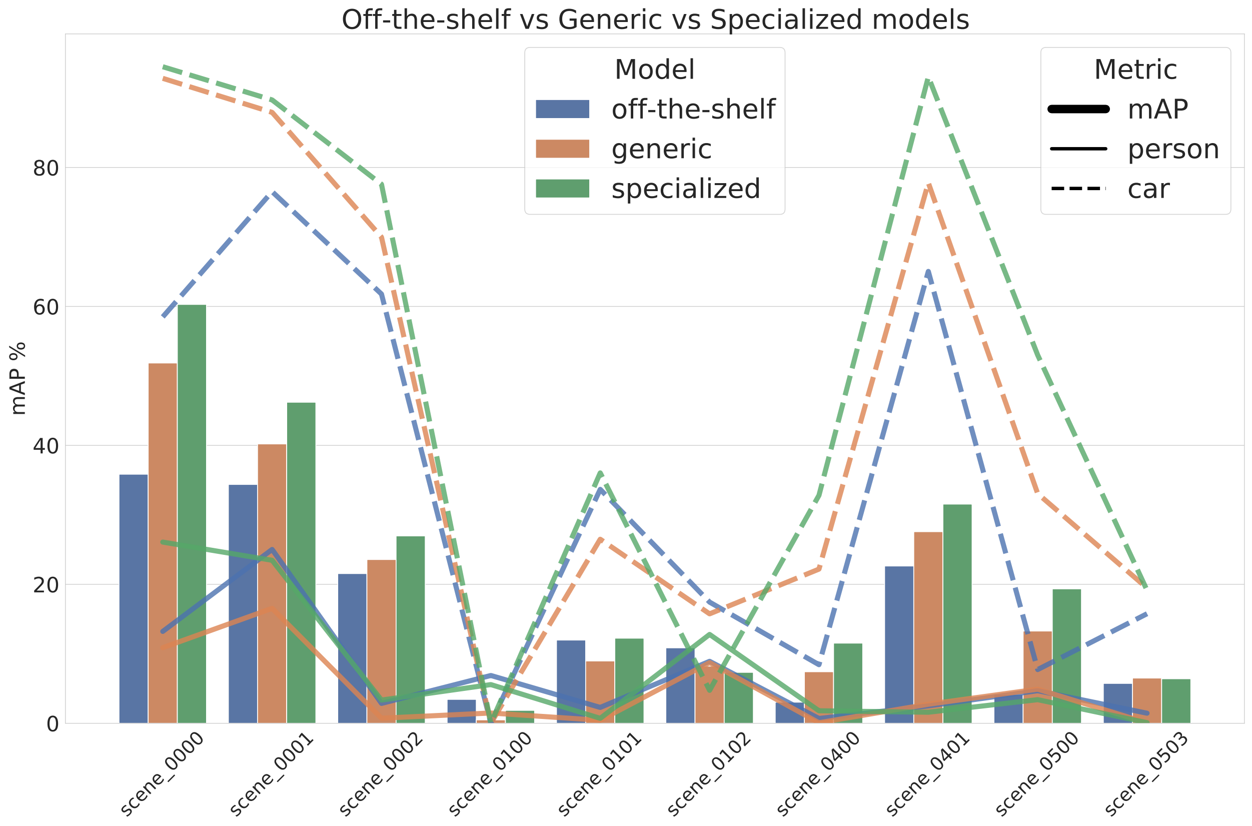


Figure 4.9: Mean Average Precision of the generic edge model compared to the specialized edge model for the VIRAT dataset. Breakdown by scene. Bars show average mAP and lines and dotted lines show person and car AP, respectively.

them during inference due to size and deficient training. However, automatic annotation successfully outperforms the results of the off-the-shelf model and increases the mAP of all scenes in average by 21%. It is worth noting that varying the confidence threshold from 0.3 to 0.5 does not result in substantial differences. This seems to be due to the low number of False Positives introduced during the annotation of the dataset.

4.7 Final Considerations

In this contribution, we have presented COVA (Contextually Optimized Video Analytics), a framework to automate the process of model specialization and ease the deployments of real-time video analytics on resource-constrained edge nodes. COVA provides a series of high-level structures and utilities that allow users to easily define the automation pipeline resulting in an object detection model specially tailored for its deployment. COVA results from an extensive exploration and a subsequent analysis of the considerations to optimize the execution of models in resource-constrained edge nodes. We have successfully identified several key characteristics of edge video analytics that allowed us to efficiently and greatly simplify the scope of the problem. For example, our results have shown that simple motion detection techniques can boost the accuracy of already trained models by allowing us to direct the inference to those parts of the frame that are moving. Moreover, COVA allowed us to explore some of these considerations, and we have shown results that can help pave the way for future improvements. For example, assuming static cameras, we have shown how it is worth assuming a certain loss of generality to boost accuracy on the specific surroundings of

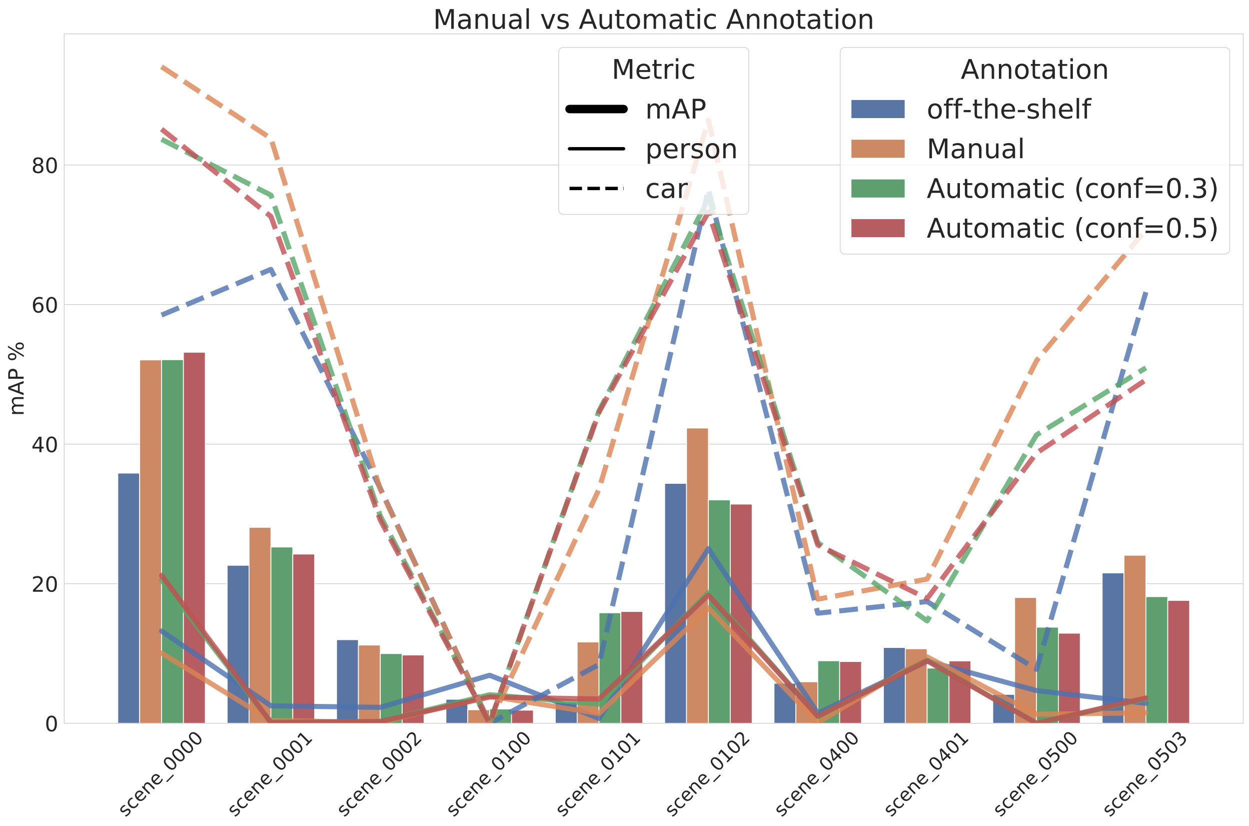


Figure 4.10: Mean Average Precision of a specialized edge model after being trained on a manually annotated dataset vs. a dataset automatically annotated by the ground-truth model with thresholds of confidence of 0.3 and 0.5. Bars show average mAP, and lines and dotted lines show person and car AP, respectively.

the deployment. Results have shown how COVA is able to fine-tune models for the context of specific cameras and increase accuracy by an average of 21% while keeping the computational cost constant and all through an automated process.

Chapter 5

Towards the Massively Distributed Edge

In Chapter 4, we demonstrated how the locality of edge deployments can be exploited by specializing models to the specific context of a camera. This allows us to provide real-time inference on resource-constrained nodes, as specialized models can be smaller than their generic counterparts while still maintaining accuracy if the problem gets like-wise reduced. In this chapter, we go one step further and present a novel method that allows the same resource-constrained nodes to work together by distributing the workload on video analytics tasks and, simultaneously, increase the amount of useful work processed by a single inference.

As the number of installed cameras grows, so does the compute resources required to process and analyze all the images captured by these cameras. Video analytics enables new use cases, such as smart cities or autonomous driving. At the same time, it urges service providers to install additional compute resources to cope with the demand while the strict latency requirements pushes compute towards the end of the network conforming a geographically distributed and heterogeneous set of compute locations, usually shared and resource constrained. Such landscape (shared and distributed locations) forces us to design new techniques that are able to optimize and distribute work among all available locations and, ideally, make compute requirements grow sublinearly with respect to the number of cameras installed. In this chapter, we present FoMO (*Focus on Moving Objects*) as the third contribution of this thesis. FoMO is a novel method to effectively optimize and distribute video analytics for static cameras on multi-camera deployments.

The chapter is organized as follows. Fist, Section 5.1 presents the literature review and previous work. Then, Section 5.2 presents and describes FoMO, a method to distribute inference by focusing on moving objects instead of full scenes. Following, Section 5.4 details the experimental setup and the methodology followed during the evaluation of FoMO, which is presented in Section 5.5. Finally, Section 5.6 concludes the chapter.

5.1 Related Work

Given the amount of data and computation required to train neural networks, the training step has garnered most of the attention from both academia and industry. However, as neural networks start being widely deployed for production use cases, optimizing inference is gaining interest from the industry. Most of the previous works has focused on the optimization of either the neural network architecture itself (using techniques to prune layers [55] or reduce weight precision through quantization) or on the optimization of the application's pipeline (using cascade classifiers [55, 43] or reducing the number of frames to process [22]). In the context of video analytics systems, there are two main directions: reducing the amount of work [22] and optimizing the inference pipeline [55].

Regarding what impacts the quality of the predictions of a neural network, the work in [21] argues that, in general, a detector can only provide high quality predictions if presented with high quality region proposals. Similarly, authors in [30] demonstrate how deep neural networks have difficulties to accurately detect smaller objects by design.

Regarding the usage of traditional computer vision techniques, authors in Authors in [19] provide a taxonomy and evaluation (in terms of accuracy and performance) of scene background initialization algorithms, which helped us decide what were the most sensible methods to implement in FoMO.

Previous methods have focused on optimizing the different transformations applied to the input image. However, all these methods still devote a significant amount of computation to decide whether a certain region contains true positives or not (region proposal) when, in practice, most will not. Our method aims to reduce the amount of work wasted on this purpose. Moreover, this allows us to stack regions of different cameras to reduce the total computation of the system and increase the overall system throughput. At the same time, our results have shown that by increasing the quality of proposals, accuracy increases considerably as a *side effect*.

To the best of our knowledge, no previous work proposes and evaluates a method combining traditional computer vision techniques to quantize and distribute portions of scene, consolidate multiple scenes into one, and process them with a single inference.

5.2 Focus on Moving Objects

Neural networks have proven to be exceptionally valuable at finding patterns from apparently unstructured data and surpass human-level accuracy on tasks like image recognition. After years of continuous research, multiple functions and mechanisms have been developed to increase accuracy of a trained model. The inductive biases, i.e., the set of assumptions that a learning algorithm uses to generalize on new samples, also impact the resulting accuracy by influencing the way the network learns patterns during training. CNNs, for example, assume certain spatial structure present in the data but our formal understanding on how this assumption translates into a given hypothesis space it is still limited.

Nonetheless, transformers [99] brought about a major breakthrough (first, in the field of Neural Language Processing (NLP) and more recently in computer vision too [27]) thanks to the use of *attention mechanisms*. Attention mechanisms try to mimic cognitive attention allowing the network to learn to focus on those parts of the input data that are more important while fading out the rest. Importance is learned during training through attention modules that highlight importance of different features. For example, spatial and channel attention are the most commonly used in computer vision and focus on spatial location and channel-wise information, respectively. Thanks to attention, the resulting models are more efficient than their CNN counterparts. However, due to fewer inductive biases, transformers require much larger amounts of training data to achieve a similar accuracy than CNNs. Thus, we can assume that inductive biases and attention during training help boost accuracy and performance during inference.

We argue that there is a window of opportunity to increase efficiency and boost accuracy during inference for CNNs analyzing images from static cameras. The analysis of static cameras allows us to mimic the effect of attention mechanisms by focus on those parts of the image we are more interested and discard the rest.

Neural networks can be trained to find objects anywhere on an image, even if they represent a small fraction of the input. To make this possible, neural networks classify many regions (in the order of thousands), proposed at an earlier stage of the network. This implies that neural networks end up looking at regions that are empty of objects of interest. We could argue that this is what makes neural networks (especially convolutional neural networks for image analysis) so powerful, as it means they can focus attention without developers needing to hint them where to look in the image. However, it also implies that most of the work will yield no meaningful results with an increased chance of wrong results (any positive detection on an *empty* region is a false positive). Moreover, neural networks have a *hard time* detecting small objects [21], i.e. small meaning they represent a small fraction of the entire scene. These small objects are, however, captured by the camera at resolutions that are often higher than that of the whole image the neural network uses as input. The problem is that once the original image is resized to match the neural network's input size, these objects often become a bunch of indistinguishable pixels. For example, an 8MP (4K UHD resolution, or 3840x2160 pixels) camera captures objects at 20 meters with a pixel density of over a 100 px/m, enough to get a sharp image of a license plate at such distance. If the image is resized to feed a network working at standard definition (640x480 pixels), density drops below 20 px/m[1], which is unsuited for most detection tasks (for reference: 320x320 pixels is a common input size for *edge* detection models).

Our work is based on the following key observation: scenes captured by static cameras do not move; objects of interest do. That is, we do not need to check the entire image if nothing new is in it. By focusing on footage from static cameras, we can *easily* extract the regions of the image that contain *objects of interest* (i.e. objects not yet classified) and filter the rest of the image out. For this task, we have multiple techniques at our disposal, like motion detection or background subtraction, that we can apply to the images captured from static cameras, extract the regions of interest, and merge them all into a composite frame that serves as input for a trained object detection model. Thus, we are able to effectively exploit the parallelism nature of neural networks to work on multiple video streams at once, using a single input image. This reduces the amount of computation that neural networks devote to analyze regions of an image that are empty of any content of interest.

5.3 The FoMO Method

In this chapter, we present FoMO (Focus on Moving Objects), a novel method to optimize and distribute video analytics workloads in edge locations. FoMO has been conceived with static cameras in mind, and its goal is to maximize the *pixel-to-object ratio* that is processed at each inference, i.e., it aims to maximize the number of pixels processed by the neural network that belong to actual objects of interest instead of the background. Towards this goal, each frame is preprocessed to extract the regions of interest that have a higher chance of containing objects of interest. Working with static cameras, we can safely assume that such regions of interest intersect with regions whose content has changed over time.

Figure 5.1 depicts the main steps involved in FoMO. First, a set of static cameras (either in a single location or geographically distributed) periodically capture images from the scenes. Then, each scene is processed individually, and background subtraction is computed to extract moving objects (i.e., regions of the image where movement has been detected). Depending on the compute and network bandwidth available at the edge, video scenes can be processed locally or sent to a central location. In both cases, a single entity, known as

[1]Calculated for a camera with 3mm focal length placed 4m height.

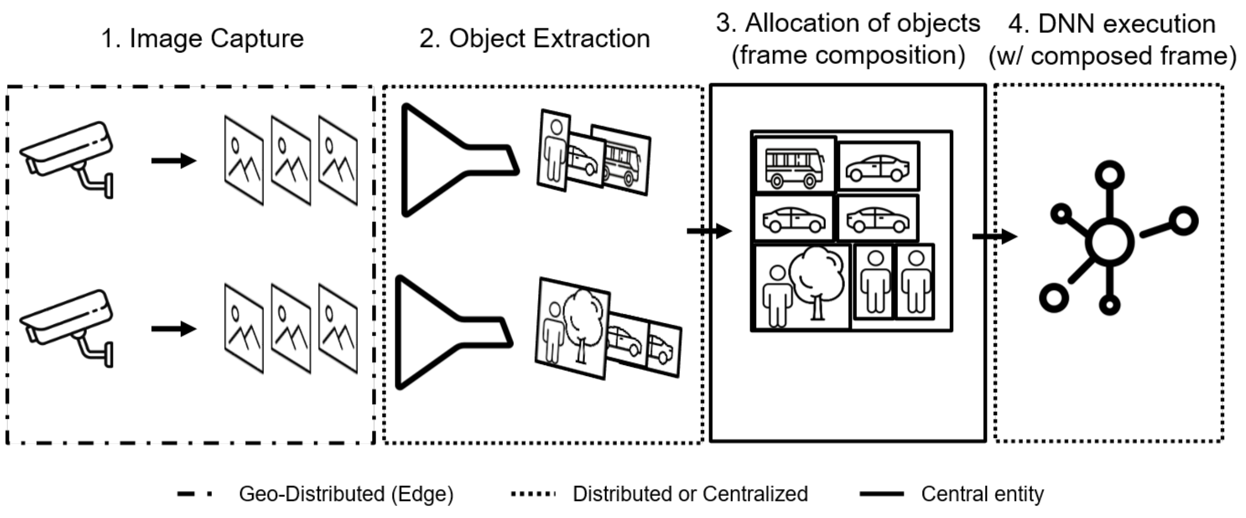

1. Image Capture
2. Object Extraction
3. Allocation of objects (frame composition)
4. DNN execution (w/ composed frame)
Geo-Distributed (Edge)
Distributed or Centralized
Central entity

Figure 5.1: Main steps involved in the process of FoMO.

composer, is in charge of receiving the objects from all the scenes and consolidating objects into a single RGB matrix known as *composite frame*. This step is called *frame composition*.

During frame composition, each object is treated as an independent unit of work that can be allocated separately or jointly from other objects from the same or other scenes. Objects are selected based on a pre-defined composition policy that determines the order and rate at which objects are composed and processed. The selected objects are placed together at each composition interval, forming a mosaic of objects (i.e., the composite frame). A few-pixels wide border is added to mark frontiers between objects. Next, the composite frame is used as input for the object detection model. Finally, the predicted coordinates of the detected objects are translated back to the original coordinates of the corresponding scene.

5.3.1 Extraction of Objects of Interest

Frame decoding results in a 3-D matrix containing the RGB color of every pixel in the scene captured by the camera. When cameras capturing the scene are static, consecutive RGB matrices will often have little to no pixel-to-pixel variation. At a higher level of abstraction, this correlation among frames means that nothing is moving in the scene (or the camera lens could not capture it). Consequently, whenever an object is moving in the scene, only some subregions are affected. This is the basis for *motion detection* algorithms, which are extensively used to trigger the analysis of a given frame to save computation on frames that do not contain anything new. In this contribution, however, we extend this idea and consider that moving objects, not whole frames, are of interest. The rationale is that for an object to trigger an action, it has to either enter, leave or interact with the scene (i.e., moving in the scene). Thus, we can focus on the regions that contain change and discard the rest. As these subregions are usually a tiny fraction of the whole frame, we can combine several subregions from different cameras into a composed frame. This frame can be processed by a neural network the same way it would process an entire frame.

Motion detection involves many challenges (e.g., camera jitter or variations in lighting, among others), and there is no single nor standard method that can handle all of them robustly. Moreover, it usually relies on background initialization algorithms to model the background without foreground objects. Among all available methods, we prioritized simplicity (i.e., speed) over accuracy for two reasons. First, the method must run in resource-constrained (edge) nodes and do it faster than it would take to process the whole frame by the neural network. Second, the accuracy of these methods is usually evaluated by error metrics that compare the modeled background to the ground-truth background at the pixel level. However, artifacts and other inconsistencies in the generated background do not necessarily translate into a lower recall of objects of interest or a lower accuracy of the object detection model.

We have evaluated FoMO using three different BGS methods to extract objects of interest from a scene. These methods differ by the way they model the background and, second, by how they identify foreground objects. The three BGS methods are:

- **PtP Mean** (Moving Average Pixel-to-Pixel): The background corresponds to the moving average of n frames (not necessarily consecutive) at pixel level. Objects are extracted by applying first a Gaussian blur to the background model and the current frame to reduce noise. Then, the absolute difference between both frames is computed before a binary threshold decides which level of difference constitutes change and which does not.
- **MOG2** (Mixture of Gaussians): Each pixel is modeled as a distribution over several Gaussians, instead of being a single RGB color. This method is better than the moving average at preserving the edges and is already implemented in OpenCV.
- **Hybrid**: The background is modeled as a Mixture of Gaussians (MOG2), but objects are extracted applying the same operations as in the PtP Mean to detect differences between the background and the current frame.

As output, all three methods provide a black and white mask of the same dimensions as the frames. From the mask, the bounding boxes of the objects moving are easily extracted by detecting adjacent pixels. Bounding boxes of an area smaller than a pre-defined minimum are discarded. This threshold helps to filter minor variations out. However, it also defines the method's sensitivity, and the optimal value will vary from scene to scene depending on the minimum size of objects (in pixels) to detect.

All three methods can run in the order of milliseconds in resource-constrained nodes. Performance and accuracy of these methods are analyzed in Section 5.5, where we explore their impact on the final results. In short, a more sensitive motion detector will generate false positives (i.e., non-interesting objects being extracted, like a tree with moving leaves) and, therefore, introduce more or larger regions than necessary to be composed in the following step.

Effectively, the breakdown of the scene into objects opens the door to two major optimizations. On the one hand, we can decide which regions should be processed and which can be omitted. Potentially, this reduces the amount of data moved around and processed by the detection model. Consequently, it increases the pixel-to-object ratio, as we achieve the same results while processing only those pixels that we have considered of interest. On the other hand, video analytics can now be distributed while consuming less network bandwidth, as, potentially, only a fraction of the scene must be sent over the network.

5.3.2 Composition Policies

After moving objects have been extracted from the frames, the composer has a pool of them in the form of cropped images, each with a different size and aspect ratio. At each composition interval, the composer decides what objects are to be allocated in the resulting composite frame. Currently, objects are selected in a purely first-come, first-serve manner. However, not all objects can be allocated during the subsequent allocation cycle, as there is a limit to the number of objects composed in one frame. The threshold is not a fixed value and is set upon one of two pre-defined composition policies.

To the composer, objects become the minimum unit of work to be allocated into composite frames. Therefore, we could argue that the composer treats composite frames as a type of resource, a resource that can be shared among requests (i.e., objects). At the same time, the composite frame can be considered elastic, as it can be overprovisioned by simply adding more objects to it. Nonetheless, as in any other type of shared resource, there is a point at which a higher degree of overprovisioning degrades the overall performance. Object detection models can locate and classify multiple objects within an input image and do it with a single forward pass over the network. However, there is a minimum number of pixels required for the model to successfully detect an object (closely related to the focal view and input size of the network). Similarly, the larger an object is, the better can a model detect it accurately.

During composition, the composer and the composition policy must consider the trade-off between the resource savings from consolidating more objects into a single composite frame and the accuracy drop involved. After a frame is composed, the resulting frame is resized to match the neural network's input size regardless of its original size. At the same time, adding one more object to the composition could potentially result in a larger frame, depending on whether the new object can be allocated without increasing the dimensions of the composite frame with all previous objects. The larger the resulting composite frame, the smaller its objects become after the frame is resized to feed the neural network. Consequently, the more objects allocated in a composite frame, the higher the amount of computation saved (i.e., fewer inferences per camera), but also the higher the impact on the accuracy of the model. Unfortunately, there is no known mechanism to determine beforehand where the limit is or even where the sweet spot is.

FoMO implements two policies that limit the number of objects allocated in a single composite frame, albeit the exact number differs from composition to composition. The first policy, namely *downscale limit policy*, limits the downscaling factor to which objects will be subjected after the composite frame is resized to feed the neural network's input. This is equivalent to setting an upper limit to the dimensions of the resulting composite frame. This policy prioritizes the quality of the detection results over the system's performance. The second policy, namely *elastic policy*, does not set a hard limit but a limit on the number of camera frames to consider on each composition. That is, the dimensions of the composite frame can be arbitrarily large but the objects allocated belong to, at most, n camera frames, being n user-defined. Effectively, this policy prioritizes system's performance over quality of the detection results.

5.3.3 Allocation Heuristic

Composing objects into a composite frame can be seen as allocating 2D images into a larger 2D canvas. For the sake of simplicity, we consider objects to be 2-D during allocation, as the third dimension is always of size 3 in RGB images. The allocation can be mapped to the *bin packing problem*. Bin packing is an optimization problem that tries to pack (allocate) items (objects) of different volumes into bins of a fixed volume (composite frame) while minimizing the number of bins used (blank spaces in the composite node). This problem is, unfortunately, known to be an NP-hard problem. Therefore, FoMO approximates a solution using a *first-fit* heuristic, where objects are first sorted into decreasing width (arbitrary) order. For this task, a sub-optimal solution results in a composite frame with more blank spaces than needed. Blank spaces reduce the density of meaningful pixels (i.e., pixels that are part of an object of interest) and, ultimately, reduce the number of objects that a single composed frame can fit.

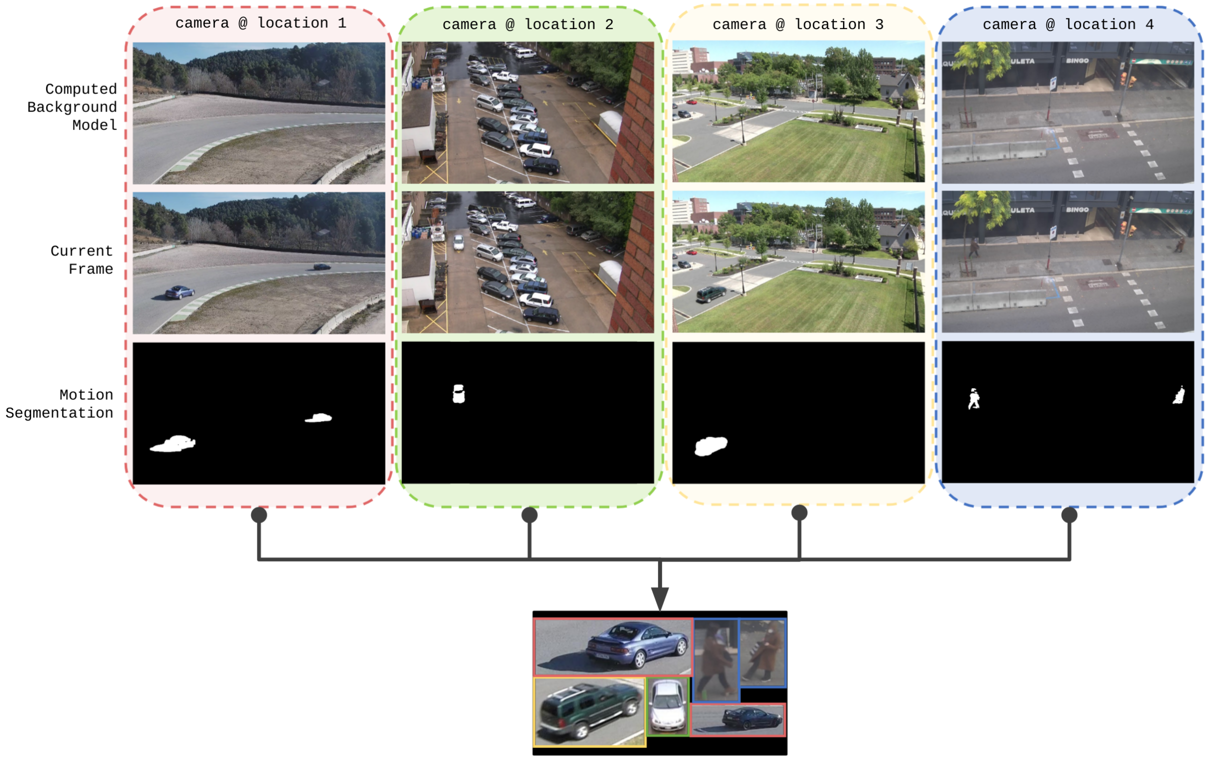


Figure 5.2: Overview of the frame composition process from 4 different cameras.

Figure 5.2 depicts the process to compose a frame from objects from four cameras. Once we obtain the moving objects from each scene, all these objects are placed into a single composed frame. As mentioned, the composition results in some blank spaces. However, objects appear much larger than they did in their scenes of origin.

5.4 Experimental Setup

In this section, we provide the specific experimental setup used throughout the evaluation presented in Section 5.5.

5.4.1 Dataset

For the evaluation, we have used the VIRAT dataset [65]. VIRAT contains footage captured with static cameras from 11 different outdoor scenes.

Dataset Curation

VIRAT contains frame-by-frame annotations for all objects in the scene, with bounding box and label (classes person, car, vehicle, bike, and object). However, annotations include both static and moving objects. We evaluate FoMO by its capability to detect moving objects. Therefore, we have curated the dataset to remove all annotations from static objects and avoid these objects artificially lowering the final accuracy. Thus, we consider objects static if their coordinates in a given frame do not change to 10 frames prior (10 has been arbitrarily chosen and corresponds to the frame skipping we used during the evaluation). Nonetheless, cameras often suffer from a slight jitter, mainly due to wind. Whenever that happens, the bounding boxes of a given object on consecutive frames may not perfectly match, even if the object has not moved. To avoid false positives in such cases, we still consider an object static when its coordinates remained static for at least 90% of the frames.

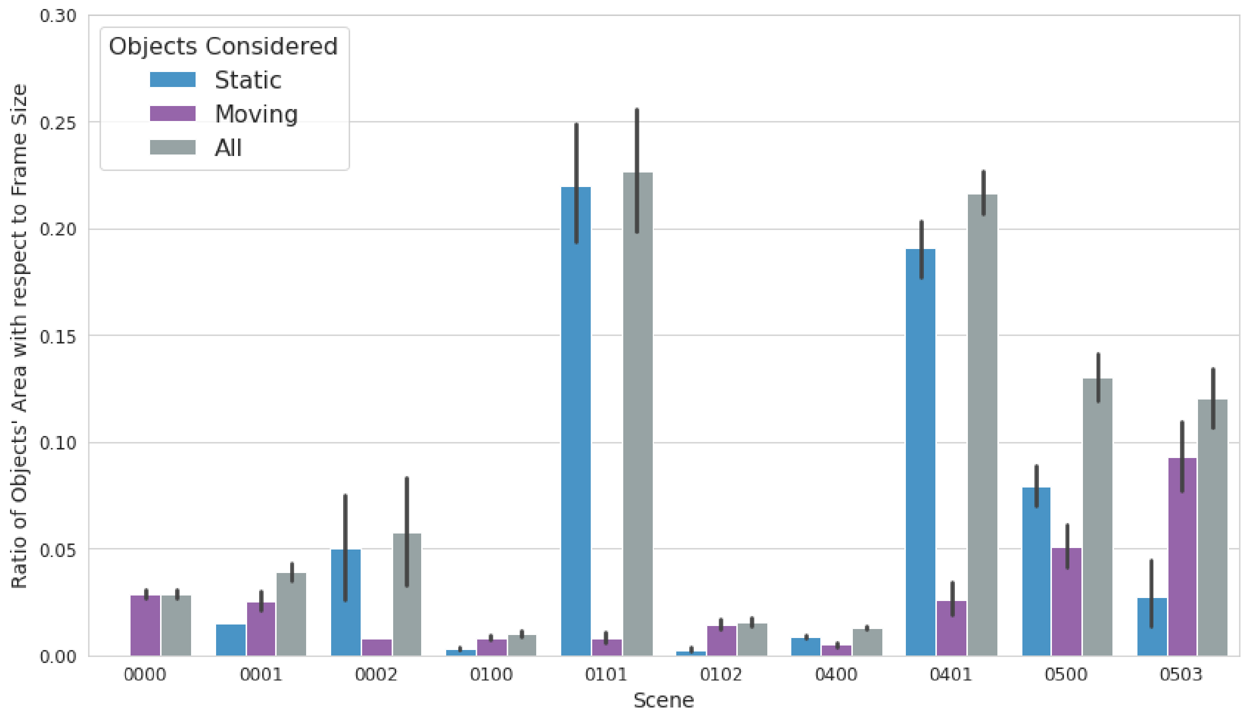


Figure 5.3: Percentage of area that represent objects in each scene, according to ground-truth annotations.

Figure 5.3 shows the percentage of area occupied by objects in each scene of the VIRAT dataset. Results show that scenes are mostly *empty* and objects represent as little as 2% of the scene and no more than a quarter of the scene. Moreover, some scenes appear to be *quiet*, i.e., only a tiny fraction of objects in the scene are moving. When considering moving objects only, the average area taken by these objects fluctuates between 1% and 8% of the scenes. These results highlight the potential benefits that can be achieved by removing all the empty regions.

Moreover, we have made the following considerations during the evaluation:

- Only sequences with at least 1000 frames have been considered.
- Discarded the first 250 frames (approx. 10 seconds) to give a time window to initialize the background modeling of some methods.
- Frame skipping of 10 frames.

5.4.2 Object Detection Models

All experiments have been carried out using pre-trained object detection models that are publicly and freely available in the TensorFlow Model Zoo [94]. We decided upon the use of pre-trained models instead of training or fine-tuning the models on our data to remove training as a variable on which to focus, which is an important one nonetheless.

The following are the three DNN models used throughout the evaluation:

- SSD + MobileNet V2 with an input layer of 320x320x3 pixels.
- Same as previous with an additional FPN (Feature Pyramid Network) that improves the quality of predictions [59].
- Same as previous with an input layer of 640x640x3 pixels to improve the quality of predictions.

5.4.3 Background Subtraction Mechanisms

Background subtraction (BGS) algorithms, albeit not directly part of our contributions, are key to extracting the set of objects that will constitute the system's workload. At the same time, they have an undeniable impact on the quality of the region proposal. An accurate BGS algorithm will extract all or most objects of interest and nothing else (i.e., discard all regions that do not contain objects of interest). However, to have a complete overview of how FoMO performs, we must break the accuracy of these methods down into *precision* and *recall*, as each one will have a different impact on the overall performance.

Precision is defined as the ratio of *True Positives* (TP) with respect to the sum of TP and *False Positives* (FP), i.e., what proportion of predictions are correct. In the context of *object extraction*, we can see precision at the pixel level and define it as the proportion of extracted area (i.e., number of pixels) that does belong to objects of interest with respect to the total extracted area (including that without objects of interest). *Recall* is defined as the ratio of TP with respect to the total number of positive examples (TP + FN), i.e. what proportion of positive examples are correctly detected. In the context of *object extraction*, we define recall as the proportion of extracted area that does belongs to objects of interest with respect to the total area of the objects of interest in the scene (including the area of the objects not extracted). Thus, low precision BGS mechanisms will cause *uninteresting* regions to occupy space in the composed frame, potentially shrinking other regions or leaving them out of the composition (hence, delaying their processing). Low recall mechanisms will cause FoMO to directly miss predictions as entire objects were not extracted and will, therefore, not make it into the neural network's input. On the contrary, a high precision and high recall BGS mechanism will maximize FoMO's efficiency, while accuracy will be made to only depend on the neural network model chosen.

The experiments aim to quantify the impact that different methods for BGS, with more or less accuracy on the extraction of objects, have on the system's resource usage and the quality of the detections.

The configuration of the three BGS methods is, for all experiments, as follows:

- PtP Mean: The moving average is computed over the last 20 frames with a frame skipping of 10 (i.e., spanning over 200 consecutive frames in total).
- MOG: Computed every frame. Implementation from OpenCV.
- Hybrid: MOG background model is updated once every 50 frames (2 seconds). The difference with the current frame to extract objects is computed with respect to the latest background model available.

5.4.4 Metrics of Interest

In video analytics, the two main metrics of interest are inference accuracy and cost. For the cost, we use latency as a proxy. For the inference accuracy, we use the average precision as used in the PASCAL Visual Object Classes (VOC) Challenge [32]. To consider a detection as either positive or negative, we use a value of 0.3 for the Intersection over Union (IoU) between the predicted and the ground-truth bounding boxes. The IoU is a matric that measures the overlap between two bounding boxes.

5.4.5 Source Code Availability

FoMO has been implemented and tested under the COVA project. As such, the source code to reproduce FoMO and the experiments described in this chapter can be found in the public repository of the COVA project [77].

5.5 Results

The following experiments evaluate the potential of our method by analyzing upper and lower-bound scenarios. Therefore, we have replicated the same stream whenever more than one stream is used unless otherwise stated. By replicating the stream, we avoid potential variability introduced by different load distributions across streams that is difficult to quantify. Consequently, if one object is captured at the *i-th* frame of a stream, the same object will be captured in the same *i-th* frame in all other streams. However, each frame's decoding and preprocessing (i.e., background subtraction, motion detection, and cropping) is computed for each stream individually to obtain an accurate and realistic performance evaluation.

5.5.1 Accuracy Boost vs Resource Savings

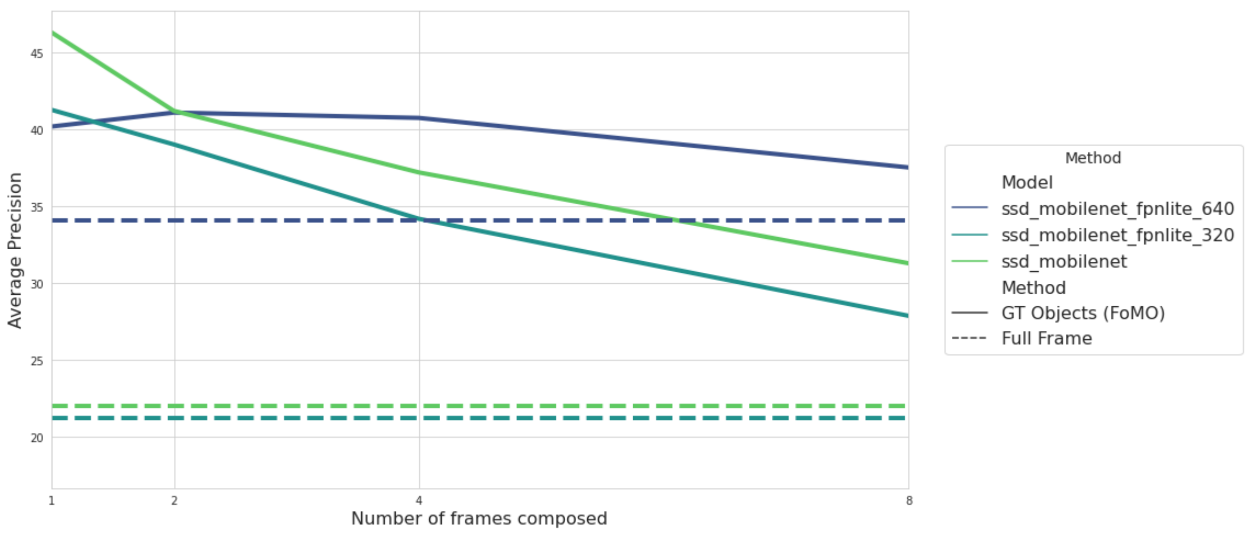


Figure 5.4: Mean Average Precision (mAP) by scene for three different DNNs with an increasing number of frames composed into a single inference. The reduction in number of inferences is equivalent to the number of frames composed.

Figure 5.4 shows the mAP averaged for all scenes of the dataset for the three different pre-trained models. Each model is evaluated using the full frame as input compared to using a composed frame with objects from 1, 2, 4, and 8 parallel streams. The objects in each frame are extracted based on the ground truth annotations from the curated dataset, i.e., without background subtraction. Therefore, these results represent an upper bound precision for FoMO. It is important to understand why the mean average precision appears especially low for the baseline method (*Full Frame*) during inference for the two smaller models. As mentioned in Section 5.4 (and, more specifically, as shown in Figure 5.3), moving objects are, on average, a tiny fraction of the frame's area. Hence, once the frame gets downscaled to the neural network's input size, objects become too small to be detected accurately. This is why the bigger model, with an input twice as large as the other two models (640x640 pixels compared to 320x320 pixels), gets around 50% improvement in accuracy over them.

From the mAP results of the models using FoMO, we extract two key observations. On the one hand, results show that a larger model does not necessarily translate into a higher mAP when inferencing over composite frames, as it happens with the baseline. The larger model is up to 10% below the smaller models when the composite frame contains objects from only one camera frame. When the number of objects to compose is small (as it is expected when only a single camera frame is considered), the composition may result in a composite frame smaller than the model's input size. Consequently, the frame must be enlarged to the appropriate dimensions expected by the neural network. Thus, these results indicate that accuracy is also impacted negatively when objects are zoomed in. On the other hand, as already discussed, accuracy takes a hit when more frames (hence, more objects too) are considered during the composition, as objects become smaller. However, results show that a larger input size seems to mitigate this impact. The largest model still achieves 10% higher accuracy composing frames eight cameras compared to the full-frame inference using the same model.

5.5.2 Impact of BGS Techniques

There are multiple available techniques to compute background subtraction to detect and extract moving objects in a scene. Each technique provides a different trade-off in terms of latency (or required resources) and quality of the solution. The quality of these techniques, however, can be measured through different metrics. We are interested in detecting moving objects but not *everything* that is moving (e.g., leaves or camera jitter). Some methods are more sensitive to minor variations than others, and the level of sensitivity will impact what is considered actual movement and what is just background noise. Therefore, a higher sensitivity potentially results in a higher rate of false positives (i.e., lower precision) and a lower rate of false negatives (i.e., higher recall), as more objects will get extracted.

Figure 5.5 shows the recall, precision, and average precision of the three object detection models using the four BGS methods considered plus the baseline (*Full Frame*, i.e., no object extraction). On the one hand, results show how FoMO performs compared to the baseline method. The upper bound for FoMO can be defined by the ground-truth annotations (*GT Objects*), as no object is missed during the extraction. GT Objects consistently outperforms the baseline by a considerable margin on every metric and model, except one case. For example, GT Objects achieves twice the AP of the baseline for the two smaller models, although this margin reduces to 30% when the larger model processes the objects. However, baseline surpasses FoMO's upper bound recall by a 10% when using the largest model. This seems to be related to what results in Section 5.5.1 showed when composition considers objects from only one frame, i.e., the resulting composite frame is smaller than the neural network's input, and objects must be enlarged. Nonetheless, a baseline's precision half of FoMO's mitigates this issue.

On the other hand, results show what can be expected from using a more or less accurate BGS method. MOG and Hybrid are close on all metrics. Again, precision is where FoMO seems to provide larger benefits, especially for small input sizes. Both MOG and Hybrid outperform baseline's precision by a factor of 2 using the smaller models and close to 50%

for the larger model. However, all BGS methods seem to miss objects during extraction, and recall takes a hit. Nevertheless, the AP of both methods still outperforms the baseline by a 60% and 30% when using the two smaller models, while baseline outperforms the other two by 12% in AP using the larger model. Finally, PtP Mean lacks far behind on the recall, which hits its average precision. Nevertheless, results show that FoMO consistently increases precision regardless of the BGS method, while the accuracy of the selected method will mainly impact its recall.

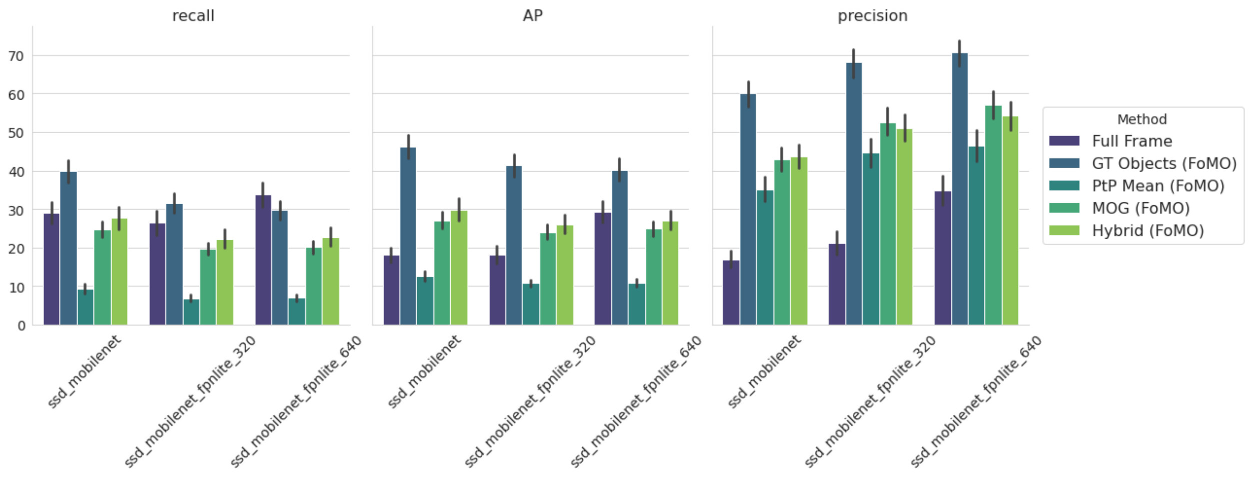


Figure 5.5: Recall, Precision, and Average Precision of three SSD MobileNet V2 pre-trained models after using the different methods to create the input for the model. *Full Frame* uses the frame as captured by the camera; GT Objects: objects are cropped from the ground truth annotations; MOG: Mixture of Gaussians; PtP Mean: Pixel-to-Pixel Mean; Hybrid: MOG for background modeling and then pixel-to-pixel difference with respect the current frame.

5.5.3 Performance and Resource Usage

FoMO's main goal is to improve the overall system performance. This is achieved by reducing the amount of computing required to process a single frame. After decoding, each frame undergoes three consecutive steps: 1. object extraction, 2. frame composition, 3. inference. We now evaluate the first two, as the third will be determined by the neural network used.

Table 5.1 shows the average latency to extract objects of interest for each of the three methods considered in the experiments. These results were obtained using a single core on an Intel Xeon 4114. Pixel-to-Pixel mean is, on average, eight times slower than MOG2 and almost six times slower than the Hybrid method. The table does not show timings for the executions that use ground truth annotations as those are synthetic executions that do not require any background subtraction to get the objects' bounding boxes.

Method	**Latency (millisec)**
PtP Mean	22.0
MOG2	2.7
Hybrid	3.8

Table 5.1: Average latency (milliseconds) to extract objects using the three methods considered in the experiments

.

The next step is the composition of objects into the composite frame. Potentially, the composition is performed once for multiple scenes. Figure 5.6 shows the time required to compose a frame with respect to the number of objects to compose and the width of the resulting composite frames (and height, as these are always 1:1). Results show how the latency increases with the number of objects to compose. The number of camera frames considered within a composition interval indirectly impacts the latency, as the more frames are considered, the more objects we can expect to be available for composition (although not necessarily always true). The increasing dimensions of the resulting composite frames highlight this relationship. The frame is enlarged to try to fit a larger number of objects. Nonetheless, we can expect the detection models to yield worse accuracy when processing the larger composite frames, as objects will appear smaller once shrunk to the model's input size. Therefore, datapoints with composition latency in hundreds of milliseconds are too large to be reliably used unless the detection model has an appropriately sized input layer. In that case, inference cost will also be higher, and, therefore, the cost of composing many objects can be hidden.

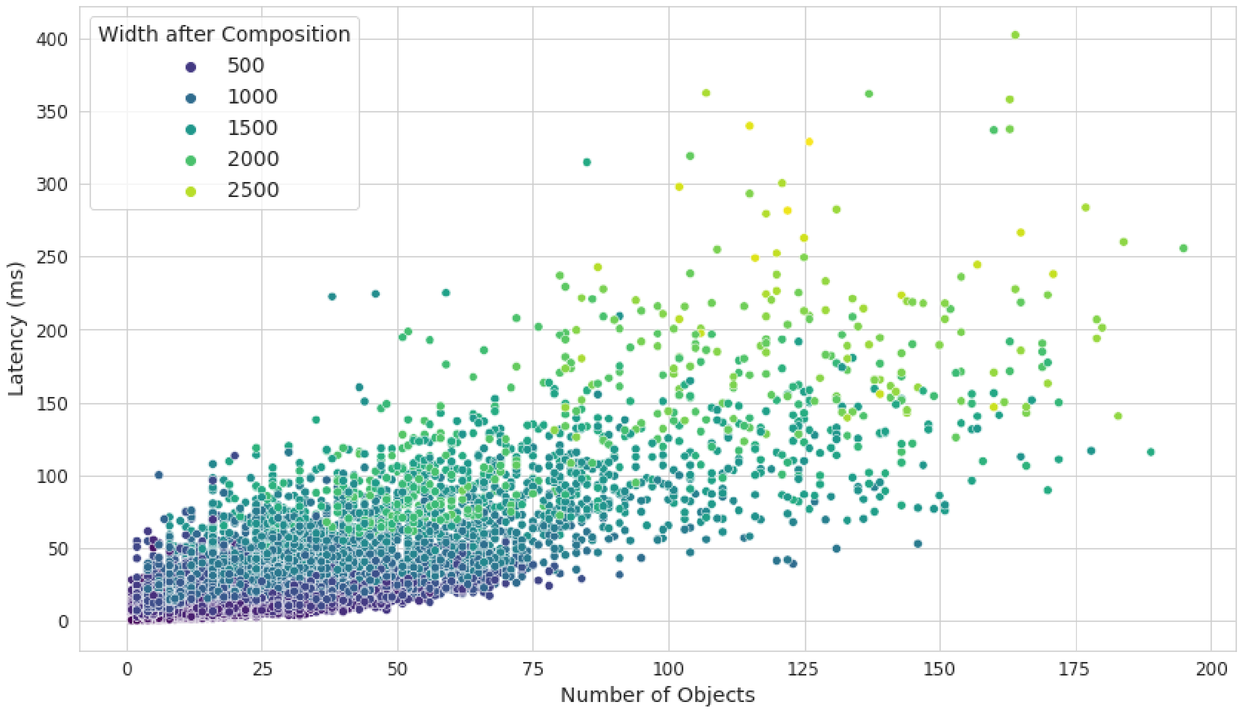


Figure 5.6: Time to compose a frame, i.e. composition latency, in milliseconds (Y-Axis) with respect to the number of objects to compose (X-Axis). The color palette shows the width of the resulting composite frame. Each dot is a composition with objects from up to eight parallel streams.

5.6 Final Considerations

In this chapter, we have presented FoMO (Focus on Moving Objects), a novel method that allows the distribution of video analytics workloads. At the same time, FoMO reduces the compute requirements of video analytics and improves the accuracy of the neural network models used for object detection. Results have shown how FoMO is able to scale the number of cameras being processed with a sublinear increment in the amount of resources required while still improving accuracy with respect to the baseline. Results have shown that the number of inferences can be reduced by 8 while still achieving between 10% and 40% higher mean average precision with the same model. Moreover, the methodology used shows how video analytics can be effectively deployed in resource-constrained edge locations by tackling its optimization from a different perspective that opens a new line of research.

Chapter 6

Conclusion and Future Work

6.1 Conclusions

The purpose of this thesis was to explore and find novel ways to optimize the deployments of video analytics applications to distributed and resource-constrained edge infrastructures. We presented three contributions that are three incremental steps towards this goal. First, we defined the requirements of real-time edge video analytics through the characterization of a heterogeneous set of hardware platforms and neural networks. The second contribution explores means to automatically deploy and optimize video analytics working with edge static cameras. This exploration resulted in the COVA framework, which was presented and evaluated within the contribution's chapter. Finally, the third contribution presented a novel method to massively distribute video analytics among edge locations. Together, these contributions brings us one step closer to large-scale edge cloud deployments for video analytics.

6.1.1 Characterization of Heterogeneous Edge Platforms

The first contribution of this thesis is a complete characterization of video analytics within a highly heterogeneous context such as the edge. In the edge, heterogeneity does not only come from the different hardware platforms installed but also, and even more importantly, the wide range of services to provide, locality of users, or restrictions imposed by the geographical location of the deployment itself.

Consequently, in this first contribution, we first presented a taxonomy of the different edge locations and how these map into different sets of hardware platforms that are expected to populate said locations. This taxonomy served as the starting point from which select the hardware platforms (including media and neural network accelerators) to cover a wide range of edge configurations. Moreover, the heterogeneity has been extended to an equally heterogeneous set of neural networks covering different video analytics use cases.

Once the experimental environment had been set, we characterized end-to-end real-time video analytics by analyzing the typical video analytics pipeline composed of a first stage of video decoding followed by the execution of neural networks. The results showed that decoding and inference are subject to different considerations and demonstrated the importance of considering both when estimating end-to-end video analytics performance and, hence, the optimal hardware for a given edge deployment, as opposed to selecting platforms based solely on their inference performance as is often done. Both, decoding and inference, may become the bottleneck depending on the platform and the use case.

At the same time, the end-to-end characterization yielded interesting observations with respect to the type of compute best suited for every scenario. The results of the characterization have been used to build an experimental model to project performance, latency, and energy efficiency metrics of the different platforms when subject to different scenarios, which let us explore the trade-offs of the different platforms to select the optimal platform based on a metric to optimize. These results showed how even the most resource-constrained hardware can be the optimal platform to accelerate certain types of use cases. For example, in a location with low user aggregation, the most resource-constrained hardware may be enough to meet certain minimum throughput or latency requirements for the use case, while also being the most energy efficient. Finally, the results of the model served as a means to define the direction of the next steps.

6.1.2 Large-Scale Deployments for Edge Video Analytics

The second contribution of the thesis focused on facilitating the migration of video analytics to edge environments with limited compute resources. This was achieved in two steps.

First, we argued that, when analyzing images from static edge cameras, we can exploit locality that the edge provides to specialize the neural networks. Said specialization allowed us to improve accuracy of the lightweight models with no extra cost in compute once the model is deployed. Consequently, such lightweight models can be used for use cases that require a certain level of accuracy while running on resource-constrained nodes.

Second, we presented COVA (Contextually Optimized Video Analytics), a framework that automates the end-to-end process to specialize models for edge video analytics. COVA is the result of an extensive exploration and a subsequent analysis of the considerations to optimize the execution of models in resource-constrained edge nodes. We have successfully identified several key characteristics of edge video analytics that allowed us to effectively simplify the scope of the problem. For example, our results have shown that simple motion detection techniques can give a considerable boost in the accuracy of already trained models by allowing us to direct the inference to those parts of the frame that are moving.

Moreover, COVA allowed us to explore some of these considerations, and we have shown results that can help pave the way for future improvements. For example, assuming static cameras, we have shown how it is worth assuming a certain loss of generality to boost accuracy on the surroundings of the deployment in return. Results have shown how the framework is able to specialize models for the context of specific cameras and increase accuracy by an average of 21% while keeping the computational cost constant and all through an automated process.

6.1.3 Distributing Video Analytics in the Edge

The third contribution of this thesis presents and evaluates FoMO, a novel method to optimize video analytics workloads while enabling the distribution of work thanks to the breakdown of the scenes into different regions of interest, which can be distributed independently throughout the network. Regions from different cameras are consolidated into a single image that is passed to a neural network. Thus, FoMO reduces the total number of inferences required to analyze multiple cameras, which allows for a sublinear growth of the compute requirements with respect to the number of cameras installed.

Results have shown how FoMO offers the system the ability to scale the number of cameras processed with a sublinear increment in the resources consumed. Moreover, as a by-product of the scene breakdown, FoMO has been found to improve the accuracy of the object detection models used in the evaluation compared to the same models working over the full scene. For example, results have shown that the number of inferences can be reduced by 8 while still achieving between 10% and 40% higher mean average precision while using the same model without any further training.

As a result, FoMO shows how video analytics can be effectively deployed in resource-constrained edge locations by tackling its optimization from a different perspective that opens a new line of research, as the level of consolidation can be considered a new variable to control the compute-accuracy trade-off.

6.2 Future Work

The contributions presented in this thesis bring video analytics closer to be accelerated at large-scale by the edge cloud. However, we believe they also open several interesting topics for future research. Hereby, we present some of those we believe are the most promising directions to take following this thesis.

6.2.1 Extend contributions to other edge use cases

In this thesis, we focused our contributions on video analytics due it to being one of the primary use cases to be deployed in the edge but also due to its complexity to run on resource-constrained nodes. However, the edge cloud will be composed by a myriad of different use cases and applications and some of those require especial attention due to their scale, like those involved in processing data generated by billions of IoT devices.

6.2.2 Contextual models

In the second contribution of this thesis, we presented the COVA framework that provides methods for automating the tuning and specialization of object detection models. However, we believe there is room to improve the automated analysis to make it more robust by using other features that can be easily extracted from the context, such as time of the day.

Nonetheless, it is worth stating that the edge model used throughout the experiments, even if arguably small, accounts for the non-negligible amount of 4.5 billion trainable parameters. It uses a MobileNetV2 architecture that can achieve a considerable 72% accuracy on the notoriously generalist ImageNet and 19% mAP on COCO when coupled with a Single Shot Detector head for box regression. Such neural network architectures can be useful, especially if they have been pre-trained on a related problem (even if it is loosely related). They offer a reasonable level of generalization that makes it easier to adapt to new environments, all at a relatively low computational cost.

However, we believe there is a margin to further specialize the models used while keeping the same accuracy. Nevertheless, it is important to highlight the challenges involved in specializing neural networks. The more specialized a model is, the harder it is to use it reliably under uncontrolled environments because we are effectively increasing the number of possible inputs for which the model cannot generalize. Analogously, the better the automated analysis of a camera's context, the better and faster we can detect a concept drift, i.e., detect whenever the scene's context turns into something the model is not trained to recognize.

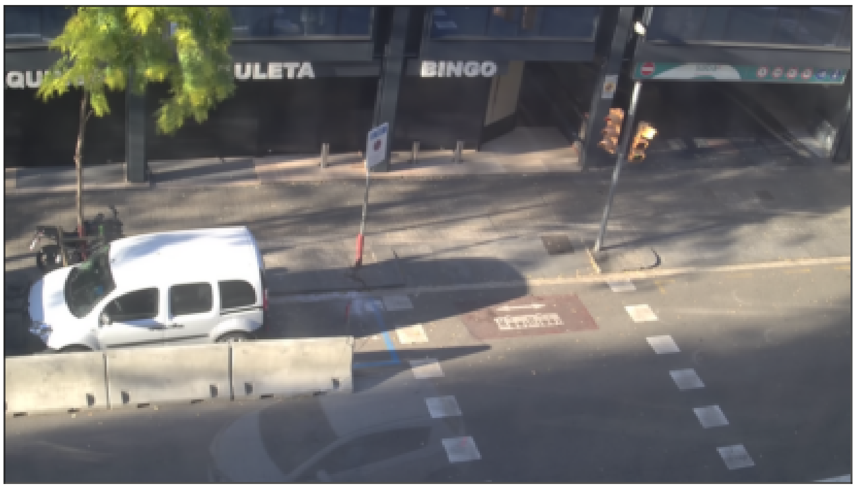
Consequently, as future work, we intend to explore alternative methods to further exploit the context of a camera. The modular design of COVA and its components open the door to seamlessly introduce new contextual features into the analysis of the scene, during the stages of automatic annotation and generation of the training dataset. We expect these new features to drive the generation of inexpensive and highly-specialized models that can still achieve high accuracy. Towards this goal, we intend to explore the use of custom and highly specialized neural networks for image classification and rely on the same techniques already used for motion detection to leverage localization. These networks are orders of magnitude smaller than the one used as the edge model previously mentioned. As a result, they are unable to generalize even in the slightest. In contrast, several of these can be trained for each camera context due to their small size and be quickly re-trained as soon as a concept drift is detected.

We could argue that COVA leverages the context of a camera by exploiting the implicit inductive biases of convolutional neural networks, i.e., CNNs learn to consider the surroundings of an object and COVA uses during training images where objects are seen in the same context they will be when captured during inference. However, the context can offer much more than just visual information. For example, when considering static cameras within a given context, objects tend to follow pre-established paths unless an extraordinary event occurs. From the point of view of a street camera, cars drive following the driveway, while pedestrians walk mostly on the sides of the street and the crosswalk. Therefore, the coordinates of an object can be indicative of the type of object. Additionally, if the camera is placed in the middle of the street, cars and pedestrians get bigger or smaller as they get closer or further from the camera while following their paths. Consequently, the size of an object, along with its coordinates, can tell us its distance from the camera and, thus, be indicative of the type of object. Figure 6.1 depicts an example of how the type of object is strongly related to its position and size within the scene where it is captured. In the example, pedestrians (blue) walk mainly on the sidewalk and the crosswalk. Similarly, cars (orange) and motorcycles (green) are mostly captured on the driveway or entering the parking lot. Pedestrians and cars can be seen together in the region of the crosswalk, but the fact that not many people are the size of a car helps discern between them. Therefore, we observe a strong relationship between the position and size of an object with the type of object. It is important to highlight that the distribution shown in Figure 6.1b will vary even if the camera is only shifted in place, let alone moved to a different location. Therefore, these features can be considered part of a camera's context, and context is tightly coupled to its camera. It is precisely for this reason that the process of model specialization must be automated or, otherwise, it becomes unfeasible at scale.

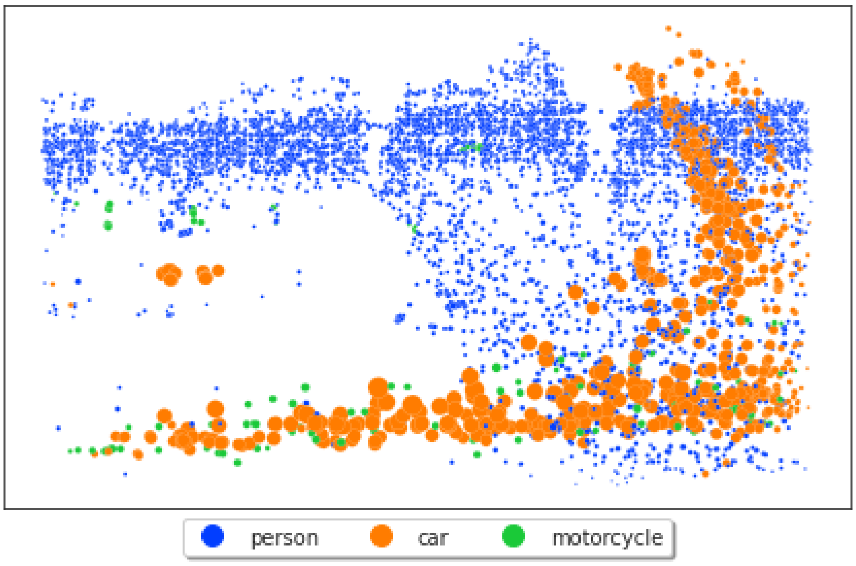
6.2.3 Characterizing FoMO's Compute-Accuracy trade-off

Other than allowing us to distribute and consolidate requests on video analytics workloads, results have shown how the use of FoMO manages to increase accuracy of the model. The accuracy boost, however, is highly dependent on the number of requests or objects composed in a single inference request. As the level of consolidation increases (and, thus, efficiency) the resulting accuracy starts to drop until it yields worse results than those compared to process full frames. Therefore, through FoMO, we obtain a useful trade-off of speed vs accuracy. Results suggest that the bigger the input of the model, the higher the consolidation it can take before accuracy drops below baseline, even after adjusting by the increase in the input size.

Nonetheless, if said trade-off could be characterized, it could be effectively used to implement orchestration policies that can adapt the consolidation level in real-time to optimize efficiency while making sure SLAs are still met.



(a)



(b)

Figure 6.1: (a) Image captured by a static camera on a street. (b) Position (within the scene) and size of the objects captured by the camera throughout a day. Coordinates represent the centroid of the object and the point size is relative to the object size as per its detected bounding box.

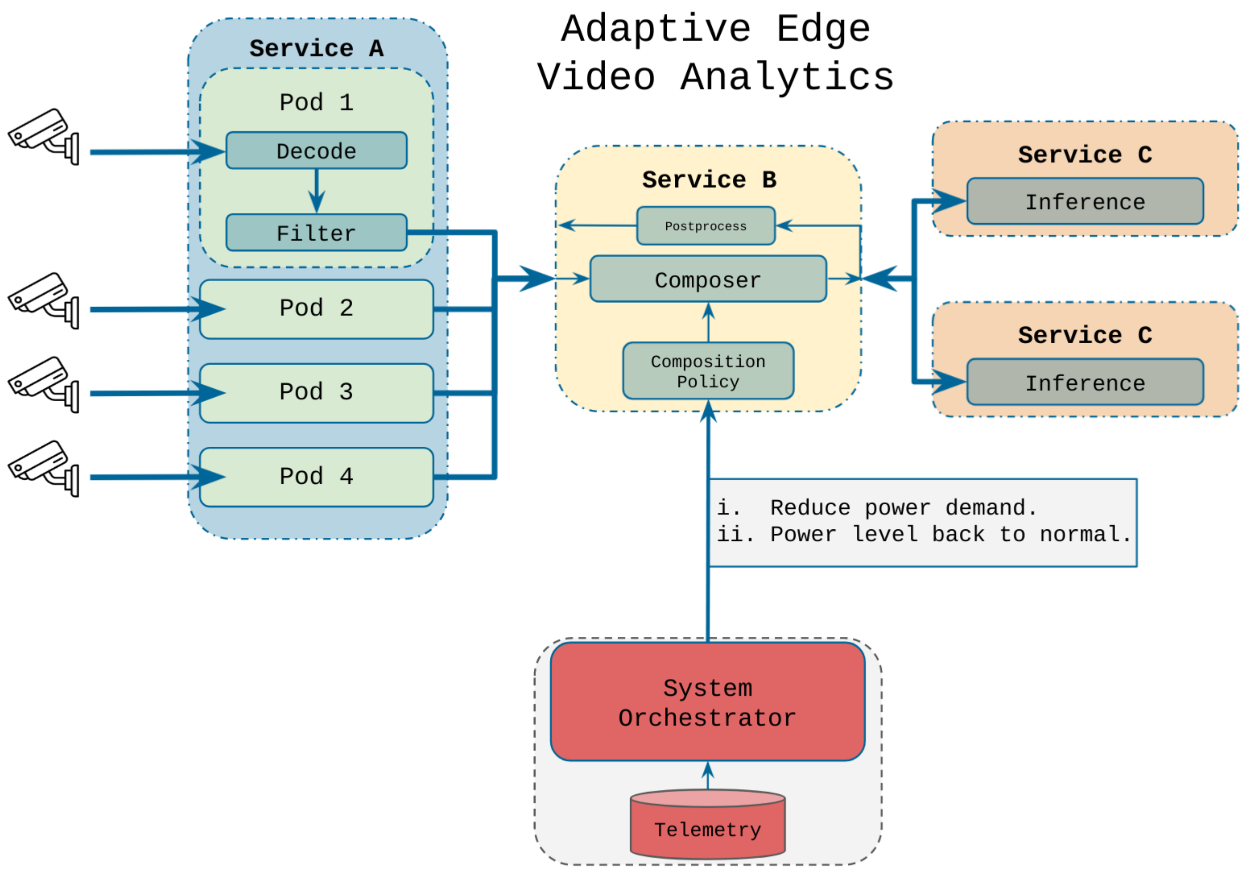


Figure 6.2: Micro-services architecture implemented using FoMO to dynamically adjust the consolidation level and, thus, power consumption based on telemetry.

6.2.4 Adaptive Power-Accuracy Policies

Other than allowing us to distribute and consolidate requests on video analytics workloads, results have shown how the simple use of FoMO manages to increase accuracy of the model. The accuracy boost, however, is highly dependent on the number of requests or objects composed in a single inference request. As the level of consolidation increases (and, thus, efficiency) the resulting accuracy starts to drop until it yields worse results than those compared to process full frames. Therefore, through FoMO, we obtain a useful trade-off of speed vs accuracy. This opens the door to build systems that can automatically adapt the level of consolidation to external factors that may limit performance. Thus, FoMO allows us the software to meet existing Software-Level Objectives (SLOs) even if the orchestration layer needs to do power capping. Figure 6.2 depicts an example of the micro-services architecture that we implemented using FoMO. It is composed of three services. *Service A* decodes incoming streams and filters its contents using motion detection. Moving objects are extracted and sent to *Service B*, which implements the logic for the adaptive composition. Every frame composed is sent to *Service C* to perform inference. Results are sent back to *Service B*, who will translate coordinates back to the original camera coordinates. Moreover, *Service B* exposes an API that allows the resource orchestration layer to set the composition policy into power-saving or normal.

Bibliography

[1] EAI Edge-IoT 2021. Best Paper Award. https://edge-iot.eai-conferences.org/2021/best-paper-award/. [Accessed: 2021-12-20].

[2] Ejaz Ahmed and Mubashir Husain Rehmani. Mobile edge computing: opportunities, solutions, and challenges, 2017.

[3] Abdelkader Aissioui, Adlen Ksentini, Abdelhak Mourad Gueroui, and Tarik Taleb. On enabling 5g automotive systems using follow me edge-cloud concept. *IEEE Transactions on Vehicular Technology*, 67(6):5302–5316, 2018.

[4] Muhammad Ali, Ashiq Anjum, M Usman Yaseen, A Reza Zamani, Daniel Balouek-Thomert, Omer Rana, and Manish Parashar. Edge enhanced deep learning system for large-scale video stream analytics. In *2018 IEEE 2nd International Conference on Fog and Edge Computing (ICFEC)*, pages 1–10. IEEE, 2018.

[5] Amazon. Amazon Elastic Graphics. https://aws.amazon.com/ec2/elastic-graphics/. [Accessed: 2022-01-10].

[6] Ganesh Ananthanarayanan, Paramvir Bahl, Peter Bodík, Krishna Chintalapudi, Matthai Philipose, Lenin Ravindranath, and Sudipta Sinha. Real-time video analytics: The killer app for edge computing. *computer*, 50(10):58–67, 2017.

[7] Michael R Anderson, Michael Cafarella, German Ros, and Thomas F Wenisch. Physical representation-based predicate optimization for a visual analytics database. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 1466–1477. IEEE, 2019.

[8] Mattia Antonini, Tran Huy Vu, Chulhong Min, Alessandro Montanari, Akhil Mathur, and Fahim Kawsar. Resource Characterisation of Personal-Scale Sensing Models on Edge Accelerators. In *Proceedings of the First International Workshop on Challenges in Artificial Intelligence and Machine Learning for Internet of Things*, pages 49–55, 2019.

[9] Gagangeet Singh Aujla, Rajat Chaudhary, Kuljeet Kaur, Sahil Garg, Neeraj Kumar, and Rajiv Ranjan. Safe: Sdn-assisted framework for edge–cloud interplay in secure healthcare ecosystem. *IEEE Transactions on Industrial Informatics*, 15(1):469–480, 2018.

[10] AWS. Global Infrastructure Regions & AZ. https://aws.amazon.com/about-aws/global-infrastructure/regions_az/, 2021. [Accessed: 2021-07-15].

[11] Microsoft Azure. Choose the Right Azure for You. https://azure.microsoft.com/en-us/global-infrastructure/geographies/, 2021. [Accessed: 2021-07-15].

[12] Ejder Bastug, Mehdi Bennis, Muriel Médard, and Mérouane Debbah. Toward interconnected virtual reality: Opportunities, challenges, and enablers. *IEEE Communications Magazine*, 55(6):110–117, 2017.

[13] Sara Beery, Grant Van Horn, and Pietro Perona. Recognition in terra incognita. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 456–473, 2018.

[14] Sara Beery, Guanhang Wu, Vivek Rathod, Ronny Votel, and Jonathan Huang. Context r-cnn: Long term temporal context for per-camera object detection. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13075–13085, 2020.

[15] Yannick Benezeth, Pierre-Marc Jodoin, Bruno Emile, Hélène Laurent, and Christophe Rosenberger. Comparative study of background subtraction algorithms. *Journal of Electronic Imaging*, 19(3):033003, 2010.

[16] Simone Bianco, Remi Cadene, Luigi Celona, and Paolo Napoletano. Benchmark analysis of representative deep neural network architectures. *IEEE Access*, 6:64270–64277, 2018.

[17] Kashif Bilal and Aiman Erbad. Edge computing for interactive media and video streaming. In *2017 Second International Conference on Fog and Mobile Edge Computing (FMEC)*, pages 68–73. IEEE, 2017.

[18] Thierry Bouwmans, Sajid Javed, Maryam Sultana, and Soon Ki Jung. Deep neural network concepts for background subtraction: A systematic review and comparative evaluation. *Neural Networks*, 117:8–66, 2019.

[19] Thierry Bouwmans, Lucia Maddalena, and Alfredo Petrosino. Scene background initialization: A taxonomy. *Pattern Recognition Letters*, 96:3–11, 2017.

[20] Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. Once-for-all: Train one network and specialize it for efficient deployment. *arXiv preprint arXiv:1908.09791*, 2019.

[21] Zhaowei Cai and Nuno Vasconcelos. Cascade r-cnn: Delving into high quality object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6154–6162, 2018.

[22] Christopher Canel, Thomas Kim, Giulio Zhou, Conglong Li, Hyeontaek Lim, David G Andersen, Michael Kaminsky, and Subramanya R Dulloor. Scaling video analytics on constrained edge nodes. *arXiv preprint arXiv:1905.13536*, 2019.

[23] Cisco. Cisco Visual Networking Index: Forecast and Trends, 2017–2022. `https://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white-paper-c11-741490.html`, February 2017. [Posted: 2017-02-27, Accessed: 2019-10-22].

[24] Coco (common objects in context) explorer. `https://cocodataset.org/#explore`. [Accessed: 2021-08-20].

[25] Rodrigo Caye Daudt, Bertr Le Saux, and Alexandre Boulch. Fully convolutional siamese networks for change detection. In *2018 25th IEEE International Conference on Image Processing (ICIP)*, pages 4063–4067. IEEE, 2018.

[26] Darcy DiNucci. Design & new media: Fragmented future-web development faces a process of mitosis, mutation, and natural selection. *PRINT-NEW YORK-*, 53:32–35, 1999.

[27] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, et al. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[28] Anton Dries and Ulrich Rückert. Adaptive concept drift detection. *Statistical Analysis and Data Mining: The ASA Data Science Journal*, 2(5-6):311–327, 2009.

[29] José Duato, Antonio J Pena, Federico Silla, Rafael Mayo, and Enrique S Quintana-Ortí. rcuda: Reducing the number of gpu-based accelerators in high performance clusters. In *High Performance Computing and Simulation (HPCS), 2010 International Conference on*, pages 224–231. IEEE, 2010.

[30] Christian Eggert, Stephan Brehm, Anton Winschel, Dan Zecha, and Rainer Lienhart. A closer look: Small object detection in faster r-cnn. In *2017 IEEE international conference on multimedia and expo (ICME)*, pages 421–426. IEEE, 2017.

[31] Christian Esposito, Aniello Castiglione, Florin Pop, and Kim-Kwang Raymond Choo. Challenges of connecting edge and cloud computing: A security and forensic perspective. *IEEE Cloud Computing*, 4(2):13–17, 2017.

[32] Mark Everingham, Luc Van Gool, Christopher KI Williams, John Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge. *International journal of computer vision*, 88(2):303–338, 2010.

[33] Ana Juan Ferrer, Joan Manuel Marquès, and Josep Jorba. Towards the decentralised cloud: Survey on approaches and challenges for mobile, ad hoc, and edge computing. *ACM Computing Surveys (CSUR)*, 51(6):1–36, 2019.

[34] Francesc Guim Bernat, Karthik Kumar, Thomas Willhalm, Daniel Rivas Barragan, Patrick Lu. System, apparatus and method for multi-kernel performance monitoring in a field programmable gate array, March 26, 2019 2019. US Patent 10,241,885.

[35] Francesc Guim Bernat, Thomas Willhalm, Karthik Kumar, Daniel Rivas Barragan, Patrick Lu. Device, system and method to access a shared memory with field-programmable gate array circuitry without first storing data to computer node, April 7, 2020 2020. US Patent 10,613,999.

[36] Robert M French. Catastrophic forgetting in connectionist networks. *Trends in cognitive sciences*, 3(4):128–135, 1999.

[37] Dimitrios Georgakopoulos, Prem Prakash Jayaraman, Maria Fazia, Massimo Villari, and Rajiv Ranjan. Internet of things and edge cloud computing roadmap for manufacturing. *IEEE Cloud Computing*, 3(4):66–73, 2016.

[38] Jianping Gou, Baosheng Yu, Stephen J Maybank, and Dacheng Tao. Knowledge distillation: A survey. *International Journal of Computer Vision*, 129(6):1789–1819, 2021.

[39] Priya Goyal, Mathilde Caron, Benjamin Lefaudeux, Min Xu, Pengchao Wang, Vivek Pai, Mannat Singh, Vitaliy Liptchinsky, Ishan Misra, Armand Joulin, and Piotr Bojanowski. Self-supervised pretraining of visual features in the wild, 2021.

[40] 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.

[41] HiEST. Tools for platform characterization. `https://github.com/HiEST/py-sysmon`. [Accessed: 2020-09-25].

[42] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the Knowledge in a Neural Network. *arXiv:1503.02531*, 2015.

[43] Kevin Hsieh, Ganesh Ananthanarayanan, Peter Bodik, Shivaram Venkataraman, Paramvir Bahl, Matthai Philipose, Phillip B Gibbons, and Onur Mutlu. Focus: Querying large video datasets with low latency and low cost. In *13th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 18)*, pages 269–286, 2018.

[44] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7310–7311, 2017.

[45] Chien-Chun Hung, Ganesh Ananthanarayanan, Peter Bodik, Leana Golubchik, Minlan Yu, Paramvir Bahl, and Matthai Philipose. Videoedge: Processing camera streams using hierarchical clusters. In *2018 IEEE/ACM Symposium on Edge Computing (SEC)*, pages 115–131. IEEE, 2018.

[46] Imagenet large scale visual recognition challenge (ilsvrc). `http://image-net.org/challenges/LSVRC/`. [Accessed: 2019-06-19].

[47] Intel. FPGA vs. GPU for Deep Learning. `https://www.intel.com/content/www/us/en/artificial-intelligence/programmable/fpga-gpu.html`. [Accessed: 2022-01-10].

[48] Intel. Intel Movidius Myriad X Vision Processing Unit. `https://www.intel.com/content/www/us/en/products/processors/movidius-vpu/movidius-myriad-x.html`. [Accessed: 2020-08-07].

[49] Intel. Intel pac arria 10. `https://www.intel.com/content/www/us/en/products/details/fpga/platforms/pac/arria-10-gx.html`. Accessed: 2021-12-16.

[50] Intel. Introduction to Intel Deep Learning Boost on Second Generation Intel Xeon Scalable Processors. `https://software.intel.com/content/www/us/en/develop/articles/introduction-to-intel-deep-learning-boost-on-second-generation-intel-xeon-scalable.html`, November 2019. [Accessed: 2020-08-07].

[51] Albert Jonathan, Abhishek Chandra, and Jon Weissman. Awan: Locality-aware resource manager for geo-distributed data-intensive applications. In *2016 IEEE international conference on cloud engineering (IC2E)*, pages 32–41. IEEE, 2016.

[52] Norman P Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, Sarah Bates, Suresh Bhatia, Nan Boden, Al Borchers, et al. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, pages 1–12, 2017.

[53] Daniel Kang, Peter Bailis, and Matei Zaharia. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics, 2019.

[54] Daniel Kang, Peter Bailis, and Matei Zaharia. Challenges and Opportunities in DNN-Based Video Analytics: A Demonstration of the BlazeIt Video Query Engine. In *CIDR*, 2019.

[55] Daniel Kang, John Emmons, Firas Abuzaid, Peter Bailis, and Matei Zaharia. Noscope: optimizing neural network queries over video at scale. *arXiv:1703.02529*, 2017.

[56] Daniel Kang, Ankit Mathur, Teja Veeramacheneni, Peter Bailis, and Matei Zaharia. Jointly optimizing preprocessing and inference for DNN-based visual analytics. *arXiv:2007.13005*, 2020.

[57] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, et al. Overcoming catastrophic forgetting in neural networks. *Proceedings of the national academy of sciences*, 114(13):3521–3526, 2017.

[58] Dawei Li, Serafettin Tasci, Shalini Ghosh, Jingwen Zhu, Junting Zhang, and Larry Heck. Rilod: near real-time incremental learning for object detection at the edge. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*, pages 113–126, 2019.

[59] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[60] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pages 740–755. Springer, 2014.

[61] Shaoshan Liu, Liangkai Liu, Jie Tang, Bo Yu, Yifan Wang, and Weisong Shi. Edge computing for autonomous driving: Opportunities and challenges. *Proceedings of the IEEE*, 107(8):1697–1716, 2019.

[62] Xavier Masip-Bruin, Eva Marín-Tordera, Albert Alonso, and Jordi Garcia. Fog-to-cloud computing (f2c): The key technology enabler for dependable e-health services deployment. In *2016 Mediterranean ad hoc networking workshop (Med-Hoc-Net)*, pages 1–5. IEEE, 2016.

[63] Ravi Teja Mullapudi, Steven Chen, Keyi Zhang, Deva Ramanan, and Kayvon Fatahalian. Online model distillation for efficient video inference. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 3573–3582, 2019.

[64] NVIDIA. NVIDIA TESLA V100 GPU ARCHITECTURE. `https://images.nvidia.com/content/volta-architecture/pdf/volta-architecture-whitepaper.pdf`, August 2017. [Accessed: 2020-08-07].

[65] Sangmin Oh, Anthony Hoogs, Amitha Perera, Naresh Cuntoor, Chia-Chih Chen, Jong Taek Lee, Saurajit Mukherjee, JK Aggarwal, Hyungtae Lee, Larry Davis, et al. A large-scale benchmark dataset for event recognition in surveillance video. In *CVPR 2011*, pages 3153–3160. IEEE, 2011.

[66] OpenCV. Opencv: Tutorial background subtraction. `https://docs.opencv.org/master/d1/dc5/tutorial_background_subtraction.html`. [Accessed: 2021-03-20].

[67] OpenVINO. Open Model Zoo repository. `https://github.com/openvinotoolkit/open_model_zoo`. [Accessed: 2020-08-15].

[68] OpenVINO. OpenVINO - FPGA Plugin. `https://docs.openvino.ai/latest/openvino_docs_IE_DG_supported_plugins_FPGA.html`. [Accessed: 2022-01-10].

[69] OpenVINO Documentation. `https://docs.openvino.ai/latest/index.html`. [Accessed: 2021-11-10].

[70] Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

[71] Senthil Purushwalkam and Abhinav Gupta. Demystifying contrastive self-supervised learning: Invariances, augmentations and dataset biases, 2020.

[72] PyTorch. PyTorch Model Zoo. `https://pytorch.org/docs/stable/torchvision/models.html`. [Accessed: 2020-10-15].

[73] Kash Rangan, Alan Cooke, Justin Post, and Nat Schindler. The Cloud Wars: $100+ billion at stake. Technical report, Tech. rep., Merrill Lynch, 2008.

[74] Joseph Redmon and Ali Farhadi. YOLO9000: better, faster, stronger. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7263–7271, 2017.

[75] David Reinsel, John Gantz, and John Rydning. The digitization of the world from edge to core. *IDC White Paper*, 13, 2018.

[76] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28:91–99, 2015.

[77] Daniel Rivas. COVA - GitHub. `https://github.com/HiEST/cova-tuner`. [Accessed: 2022-01-20].

[78] Daniel Rivas, Francesc Guim, Jordà Polo, Josep Ll Berral, and David Carrera. Large-scale video analytics through object-level consolidation. *International Summit Smart City 360°*, 2021. doi: `10.1007/978-3-031-06371-8_11`.

[79] Daniel Rivas, Francesc Guim, Jordà Polo, Josep Ll. Berral, Pubudu M. Silva, and David Carrera. Towards automatic model specialization for edge video analytics. *Future Generation Computer Systems*, 134:399–413, 2022. doi: `10.1016/j.future.2022.03.039`.

[80] Daniel Rivas, Francesc Guim, Jordà Polo, and David Carrera. Performance characterization of video analytics workloads in heterogeneous edge infrastructures. *Concurrency and Computation: Practice and Experience*, page e6317, 2021. doi: `10.1002/cpe.6317`.

[81] Mahadev Satyanarayanan, Paramvir Bahl, Ramón Caceres, and Nigel Davies. The case for vm-based cloudlets in mobile computing. *IEEE pervasive Computing*, 8(4):14–23, 2009.

[82] Navuday Sharma, Maurizio Magarini, Dushantha Nalin K Jayakody, Vishal Sharma, and Jun Li. On-demand ultra-dense cloud drone networks: Opportunities, challenges and benefits. *IEEE Communications Magazine*, 56(8):85–91, 2018.

[83] Haichen Shen, Seungyeop Han, Matthai Philipose, and Arvind Krishnamurthy. Fast video classification via adaptive cascading of deep models. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3646–3654, 2017.

[84] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[85] Konstantin Shmelkov, Cordelia Schmid, and Karteek Alahari. Incremental learning of object detectors without catastrophic forgetting. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3400–3409, 2017.

[86] Ludwig Siegele. *Let it rise: A special report on corporate IT.* Economist Newspaper, 2008.

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

[88] Chris Stauffer and W Eric L Grimson. Adaptive background mixture models for real-time tracking. In *Proceedings. 1999 IEEE computer society conference on computer vision and pattern recognition (Cat. No PR00149)*, volume 2, pages 246–252. IEEE, 1999.

[89] Xiaoyi Tao, Kaoru Ota, Mianxiong Dong, Heng Qi, and Keqiu Li. Performance guaranteed computation offloading for mobile-edge cloud computing. *IEEE Wireless Communications Letters*, 6(6):774–777, 2017.

[90] TensorFlow. Efficientdet d2 768x768 pre-trained model. `http://download.tensorflow.org/models/object_detection/tf2/20200711/efficientdet_d2_coco17_tpu-32.tar.gz`. [Accessed: 2021-03-15].

[91] TensorFlow. Ssd mobilenet v2 320x320 pre-trained model. `http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_mobilenet_v2_320x320_coco17_tpu-8.tar.gz`. [Accessed: 2021-03-15].

[92] TensorFlow. `https://www.tensorflow.org/`. [Accessed: 2021-11-10].

[93] TensorFlow. Tensorflow 2 detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`. [Accessed: 2021-03-15].

[94] TensorFlow. TensorFlow 2 Detection Model Zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md`, 2021. [Online; accessed 25-June-2021].

[95] AWS Latency Test. AWS Latency Test. `https://ping.psa.fun/`. [Accessed: 2021-09-20].

[96] Liang Tong, Yong Li, and Wei Gao. A hierarchical edge cloud architecture for mobile computing. In *IEEE INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications*, pages 1–9. IEEE, 2016.

[97] International Telecommunication Union. Measuring digital development: Facts and figures 2020. 2020.

[98] Blesson Varghese, Nan Wang, Sakil Barbhuiya, Peter Kilpatrick, and Dimitrios S Nikolopoulos. Challenges and opportunities in edge computing. In *2016 IEEE International Conference on Smart Cloud (SmartCloud)*, pages 20–26. IEEE, 2016.

[99] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008, 2017.

[100] Anthony Vetro, Charilaos Christopoulos, and Huifang Sun. Video transcoding architectures and techniques: an overview. *IEEE Signal processing magazine*, 20(2):18–29, 2003.

[101] Shiyao Wang, Yucong Zhou, Junjie Yan, and Zhidong Deng. Fully motion-aware network for video object detection. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 542–557, 2018.

[102] Xin Wang, Yujia Luo, Daniel Crankshaw, Alexey Tumanov, Fisher Yu, and Joseph E Gonzalez. Idk cascades: Fast deep learning by learning not to overthink. *arXiv preprint arXiv:1706.00885*, 2017.

[103] YouTube. Costa Rica in 4K 60fps HDR (Ultra HD). `https://www.youtube.com/watch?v=LXb3EKWsInQ`. [Accessed: 2020-08-19].

[104] YouTube. Recommended upload encoding settings. `https://support.google.com/youtube/answer/1722171#zippy=%2Cbitrate`. [Accessed: 2021-09-20].

[105] Zhi Zhou, Xu Chen, En Li, Liekang Zeng, Ke Luo, and Junshan Zhang. Edge intelligence: Paving the last mile of artificial intelligence with edge computing. *Proceedings of the IEEE*, 107(8):1738–1762, 2019.