Politecnico di Milano
Dipartimento di Elettronica, Informazione e Bioingegneria
Doctoral Programme In Information Technology
Computer Science and Engineering

Artificial Intelligence and Augmented Reality for entertainment applications

Doctoral Dissertation of:
Darian Frajberg

Supervisor:
Prof. Piero Fraternali

Tutor:
Prof. Cesare Alippi

The Chair of the Doctoral Program:
Prof. Barbara Pernici

2019 – XXXII

Acknowledgements

There are several special people who I feel I am particularly indebted to for having in one way or another stood by me all along the way, and without whom this outcome would not have been possible.

First of all, I would like to thank my family for the permanent, unfaltering support they have given me not only through my studies but throughout my whole life, as well as their encouragement to always pursue my dreams regardless of how challenging they might be or even when it meant being far away from them. I am grateful to my mum, Fabiana, and my dad, Marcelo, who I owe a big part of this accomplishment. To my sister, Eliana, who is such an important part of my life. To my grandparents, who have always taken so much interest in me and are or would have been so proud of this.

I am deeply and forever thankful to Piero Fraternali, my thesis advisor, for his support, trust and ability to guide me, which have been invaluable to me not only for the development of this thesis but for my own personal and academic growth. He was a role model and his passion and hard work inspired me to do my very best.

Thanks are also due to all my friends and those people who have somehow always been present. A very special mention to Roman Fedorov, Rocio Nahime Torres, Carlo Bernaschina, Chiara Pasini and Sergio Herrera, who accompanied me during this adventure and helped me grow both professionally and personally and entrusted me with their dear friendship.

A very special gratitude goes to Matias Urbieta, indeed one of the big responsible people for persuading me to do a PhD.

I also want to extend my thanks to Alessandro Bozzon and David Crandall for their much appreciated reviews.

Abstract

OUTDOOR Augmented Reality applications are an emerging class of software systems that demand the fast identification of natural objects on mobile and embedded systems. They arise as an interesting tool to support the creation of entertainment and educational applications. Artificial Intelligence has recently exhibited superior performance in a variety of Computer Vision tasks and can lead to novel Augmented Reality solutions. Nonetheless, their execution remains challenging and requires non negligible resources for devices with hardware constraints. The goal of the research presented in this thesis is to exploit the commoditization of Artificial Intelligence methods and the forthcoming wave of low-cost mass market Augmented Reality devices, to propose methods, architectures and components to support the creation and evaluation of solutions for outdoor Augmented Reality applications efficiently executable on low-power portable devices. Specifically, the focus is set on entertainment applications that can play a fundamental role to motivate citizens to contribute for environmental crowdsourcing purposes, such as data collection. The experimental results demonstrate how Artificial Intelligence, Computer Vision and Augmented Reality can be successfully integrated for the construction of novel entertaining solutions for limited-capacity portable systems.

Contents

CHAPTER 1

Introduction

Outdoor Augmented Reality (AR) systems provide a new way of navigating and interacting with an environment by overlaying useful information directly on top of what the user sees [21]. These solutions analyze sensor readings and perceptually enrich the real-world view with contextual information to enable knowledge acquisition and exploration. The emergence of consumer low-cost wearable devices and, in particular, the proliferation of mobile devices offers bright prospects for the development and mass adoption of entertainment Augmented Reality applications, spanning areas such as gaming [161], tourism [120], cultural heritage [211] and education [14].

The recent success of Artificial Intelligence is tightly coupled to the Deep Learning (DL) breakthrough [129]. This was originated in 2012, when Krizhevsky et al. [121] participated in the Large Scale Visual Recognition Challenge (LSVRC) [187] with a Convolutional Neural Network [130] and won by a significant margin, thus empirically demonstrating the enormous potential of Deep Learning data-driven approaches. Since then, remarkable results have been achieved in many areas, such as medicine [59] [139] [173], autonomous vehicles [35], speech recognition [91] and translation [208], which are changing people's lives [125]. Computer Vision (CV) has been one of the most favored fields, even surpassing human-level performance in multiple benchmarks [59] [85] [201] [173].

The outstanding advances of the Computer Vision field and the widespread use of mobile devices offer unique opportunities for their combination towards the construction of enhanced and massive engaging Mobile Augmented Reality (MAR) systems [110]. Such integration has the potential to provide MAR systems not only the ability to project and augment the real-world, but also to analyze and understand the surrounding environment, thus providing better user experience.

Nonetheless, the development of CV-enhanced Mobile Augmented Reality systems poses several technical challenges, such as achieving high recognition accuracy and real-time performance, while dealing with multiple sensor fusion, unreliable connectivity, heterogeneous architectures, and limited resources, such as computation, storage, memory and battery.

1.1 Problem Statement

The goal of this thesis is to study the feasibility of integrating Artificial Intelligence, Computer Vision and Augmented Reality fields towards the construction and support of massive entertainment applications able to provide an improved user-experience. This thesis aims at answering such question and illustrates a use case in which the problem has been successfully addressed.

Problem Statement:

Given the commoditization of Artificial Intelligence methods and the forthcoming wave of low-cost mass market Augmented Reality devices, propose methods, architectures and components to support the creation and evaluation of solutions for outdoor Augmented Reality applications efficiently executable on low-power and heterogeneous portable devices.

We tackle the development of outdoor Mobile Augmented Reality applications featuring intelligent Computer Vision to exploit the surrounding visual context captured by the camera and thus, compensate potential projection errors caused by other typical noisy sensors (e.g. GPS, accelerometer, magnetometer). We specifically focus on the efficient deployment of Artificial Intelligence within mobile and embedded systems with limited resources and heterogeneous architectures. In contrast to traditional Augmented Reality systems purely based on location and/or motion sensors, the proposed approach requires non-trivial, realistic and automated testing, which is also covered in this work.

As the use case, we have targeted the exploration of mountains by implementing PeakLens, a real-world outdoor Augmented Reality mobile app that identifies mountain peaks and augments the view with their corresponding information. It is available for Android and has over 500k installs. This application can be used to crowdsource the collection of mountain images for environmental purposes, such as the analysis of snow coverage for water availability prediction [32] and the monitoring of plant diseases [150].

1.2 Research questions

In this section, we formulate the research questions that motivate the work of this thesis; we list the questions following the logical order in which they should be answered in order to successfully address the target problem.

Research Question 1. *Can the recent progress in Computer Vision be exploited for the improvement of Augmented Reality user experience?*

DL-enhanced Computer Vision (CV) has recently achieved outstanding results and can potentially benefit Augmented Reality applications due to its capacity to exploit the visual context captured from the camera. However, it is not obvious how to integrate a CV component into a mobile outdoor AR application, which requires the DL module to work in real-time, with limited resources, and in a cooperative manner with other device sensors. This research question is central and transversal to the whole thesis, and is thus treated along all the chapters.

Research Question 2. *How to efficiently deploy Deep Learning models on mobile and embedded systems with limited hardware resources and heterogeneous architectures?*

Ad-hoc optimised solutions for specific hardware architectures or specific silicon vendors can reach maximum performance. Nonetheless, such approaches may create scalability and maintenance issues when targeting a high number of extremely heterogeneous architectures and devices, as in the case of Android market nowadays. This research question is treated in Chapter 5 and Chapter 6.

Research Question 3. *How to support testing to cope with noisy multi-sensor mobile applications in lab conditions?*

Testing an outdoor AR app requires a realistic simulation of outdoor conditions, which depend on multiple integrated sensor streams, hard to reproduce in lab conditions. Besides, such simulation should be automated and provide both errors discovery and performance assessment. This research question is tackled in Chapter 7.

1.3 Contributions

The contributions of this thesis can be summarized as follows:

- We recall the image-to-terrain problem for geolocalization and camera orientation estimation purposes in natural environments, as defined in the relevant literature (e.g. [12] [13]). We illustrate a mountain skyline extraction pipeline that exploits a Convolutional Neural Network for evaluating the probability that pixels belong to the skyline, and a post-processing step for extracting the actual skyline from pixel probabilities. Differently from previous work, we tackle the presence of interruptions in the natural skyline to isolate the fragments that correspond to the terrain and optimize the model for on-board low-power mobile deployment.

- We describe a framework for the development of visual context aware Outdoor Mobile Augmented Reality applications created for addressing challenges such as achieving high accuracy, stability, real-time performance and functioning in spite of possible unreliable network connectivity. Moreover, we characterize the dimensions of the data used to augment the camera view of applications of this nature and discuss the problems posed by their management.

- We tackle the problem of Deep Learning model optimization and inference acceleration for devices with limited resources [124] [127] and discuss the design requirements of solutions capable of supporting a wide spectrum of device architectures. We present and release a public implementation of a framework (PolimiDL) for Deep Learning inference acceleration that improves performance on mobile devices and embedded systems without accuracy loss.
- We introduce the architecture of a framework for testing multi-sensor mobile applications and discuss the essential design decisions and rationale. Unlike prior work, which focused on the fidelity of replaying composite sensor sequences in emulated environments [179], on scalability of testing [134], or on the simulation of usage context at different levels [218], we concentrate on the specific scenario of assessing soft errors in multi-sensor mobile applications, exploiting context traces captured in the field.
- We illustrate the application of the proposed frameworks to the case study of real-time mountain peak identification and report the experience and evaluation results achieved by such real-world Augmented Reality multi-sensor mobile application (PeakLens). This required the collection and annotation of data for both training and testing purposes, definition of quality metrics, execution of the corresponding experiments under controlled conditions and assessment of users' feedback.

1.4 Structure of the thesis

The structure of the thesis follows the same logical flow presented in Section 1.2:
Chapter 2 discusses the background of the work contained in this thesis, focusing on concepts common to the remaining chapters, i.e. image-based geolocalization, Mobile Augmented Reality, Artificial Intelligence, efficient mobile Deep Learning and citizen science. This chapter also presents the main use case treated and used as validation along the thesis, which is a real-world mobile AR application for mountain exploration.
Chapter 3 introduces a novel approach for image-based geolocalization and camera orientation estimation on natural mountain environments, which is based on a robust Deep Learning mountain skyline detection and its subsequent alignment w.r.t. the terrain.
Chapter 4 presents a framework for Outdoor Mobile Augmented Reality applications featuring Computer Vision modules for the exploitation of the visual context captured by the camera. It also defines a characterization for the dimensions of heterogeneous meta-data used in this kind of applications and discusses their underlying management challenges. Finally, this chapter illustrates how the framework can be adapted to the use case application of mountain exploration.
Chapter 5 reports on the experimentation with state-of-the-art building blocks to optimize the model architecture definition presented in Chapter 3 to further enhance its suitability for mobile deployment.
Chapter 6 presents PolimiDL, a framework for the accelerated Deep Learning inference on low-power heterogeneous mobile systems and evaluates its performance by confronting it w.r.t. the state-of-the-art.

Chapter 7 presents a framework for multi-sensor mobile application testing and reports the evaluation in a case study by addressing soft errors in the real-world AR multi-sensor app use case.
Finally, **Chapter 8** concludes the thesis and proposes the future direction of the research in this area.

This thesis includes the material from the following publications, co-authored by the candidate:

- Roman Fedorov, Darian Frajberg, and Piero Fraternali. "A framework for outdoor mobile augmented reality and its application to mountain peak detection" [60].
- Darian Frajberg, Piero Fraternali, and Rocio Nahime Torres. "Convolutional neural network for pixel-wise skyline detection" [65].
- Darian Frajberg, Piero Fraternali, and Rocio Nahime Torres. "Heterogeneous information integration for mountain augmented reality mobile apps" [66].
- Rocio Nahime Torres, Darian Frajberg, Piero Fraternali, and Sergio Luis Herrera Gonzales. "Crowdsourcing Landforms for Open GIS Enrichment" [215].
- Darian Frajberg, Piero Fraternali, Rocio Nahime Torres, Carlo Bernaschina, and Roman Fedorov. "A Testing Framework for Multi-Sensor Mobile Applications" [67].
- Darian Frajberg, Carlo Bernaschina, Christian Marone, and Piero Fraternali. "Accelerating Deep Learning inference on mobile systems" [64].

CHAPTER 2

Background

This work pursues the integration of Artificial Intelligence, Computer Vision and Augmented Reality fields for the creation and support of entertainment applications on low-power portable devices. In this chapter, we provide an overview of the scientific literature that addresses related topics and we introduce the main use case adopted as validation along the thesis.

The chapter is structured as follows: Section 2.1 surveys image-based geolocalization techniques and recalls the specific problem of mountain skyline extraction suited for outdoor natural environments; Section 2.2 overviews previous work in the areas of outdoor Augmented Reality applications with a particular focus on Mobile AR solutions; Section 2.3 covers the use of Artificial Intelligence on mobile devices and embedded systems; Section 2.4 surveys the related work on mobile application testing, context simulation, and mobile multi-sensor application testing; Section 2.5 provides an overview of Citizen Science, focusing on data collection and processing applications for environmental monitoring purposes. Finally, Section 2.6 describes the use case of a real-world mobile AR application for mountain exploration.

This chapter includes material from the following publications, co-authored by the candidate: [60] [65] [66] [215] [67] [64].

2.1 Image-based geolocalozation

Image registration [234] is a classical Computer Vision problem that aims at aligning multiple images (the reference and sensed images) of the same scene, which can be taken from diverse sensors and spatio-temporal points, i.e. different cameras, view-points, scales, imaging conditions and capture moments. It is commonly

applied in fields such as medicine [90] and remote sensing [17] to align or fuse information.

Several research studies [26] [164] have proposed the use of image-based registration techniques for large scale visual geolocalization purposes, which can be categorized into global (geolocalization at planet-scale) [223], urban (geolocalization at city-scale) [16] and natural (geolocalization in natural environments) [12]. Crandall et al. [46] studied the geo-spatial distribution of a huge geo-tagged collection of Flickr images, automatically identified frequently photographed landmarks, and used such extracted knowledge to predict the location of images exploiting their visual, textual and temporal features. Weyand et al. [223] applied Deep Learning by training a CNN with Google Street View and Flickr data for the geolocalization of images in the map, based solely on their visual content. This study was able to surpass human accuracy, but still offers room for improvement due to the extreme complexity of the task and the presence of scenarios that may resemble different places located all around the world. geolocalization based on only image content requires huge amounts of distributed, pre-registered images, collected in different conditions (e.g. illumination, weather, season, traffic, etc), which makes scalable solutions difficult to achieve. Instead, Armagan et al. [8] presented a method to exploit camera frames to automatically refine the original estimated geolocalization and pose captured by GPS and orientation sensors in urban environments. They exploited Deep Learning for the segmentation of buildings in the input images, as well as for the subsequent alignment with respect to the 2.5D map of the surroundings; this technique is specific for urban scenarios but reduces the number of reference images needed for training.

2.1.1 Image-to-terrain registration

Image-to-terrain alignment techniques arise as suitable solutions for geolocalization, when dealing with images taken in natural mountain environments. Early work, such as [12] and [13], tackled the problem by computing the overlay between the skylines extracted from Digital Elevation Model (DEM) data and mountain images. Skyline extraction is a sub-problem of image-to-terrain alignment.

Heuristic methods based on edge detection work well on images taken in good conditions, but present difficulties with bad weather or occluded scenarios. All the documented methods perform skyline extraction off-line and at the server-side. Furthermore, all methods extract a continuous skyline spanning the whole image from side to side, even if some parts may not be visible and thus could worsen the quality of the image-to-terrain alignment.

2.1.2 Skyline extraction

The extraction of natural skylines in mountain environments can be used to address both geolocalization and camera orientation estimation problems. Baboud et al. [13] proposed an automatic approach exploiting edge-based heuristics, whereas Baatz et al. [12] applied sky segmentation techniques based on dynamic programming, which required manual support for challenging pictures. The latter also released the CH1 data set [193], which contains 203 images with their corresponding segmentation ground truth information (an example is shown in Figure 2.1).

Feature-based heuristic methods (e.g. based on edge detection) perform well on images taken in good conditions, but do not address bad weather and skyline occlusions adequately. In these cases, a cloud, a high voltage cable, or a roof impact negatively on the heuristic edge filter, e.g. a cloud edge would be treated as skyline and the mountain slope below it would be erroneously regarded as noise.

Figure 2.1: *Sample image from CH1 data set [193] with sky/terrain segmentation.*

The skyline extraction problem can also be addressed with Machine Learning techniques. Hung et al. [100] proposed the application of a Support Vector Machine (SVM) to predict the skyline pixels based on diverse color and edges information features, followed by a dynamic programming strategy to link the disconnected skyline fragments. Ahmad et al. [2] also applied Machine Learning to compute skyline heat maps, followed by dynamic programming, thus not considering the presence of interruptions. The authors compared the use of a SVM and a CNN trained on a very small data set (9 images taken in the same location) and as a result concluded that the SVM was able to generalize better than the CNN for the test data sets under evaluation. Porzi et al. [169] used the CH1 [193] data set to extract the skyline with a Deconvolutional Neural Network for image segmentation; their approach treats an input image as a foreground-background segmentation problem and does not single out obstacles. Alternative approaches for horizon detection were compared and tested in [3] by using the Geopose3k data set [25]. However, the Geopose3k data set annotation was produced in a semi-automatic manner and is not completely consistent, provided that the render obtained from the DEM is considered as ground truth even if its silhouette does not perfectly match the corresponding image or if part of the natural skyline is interrupted by obstacles, as shown in Figure 2.2. Differently from previous works, we annotated the skyline in images manually, taking into account the occluded fragments, as presented in Figure 2.3.

Object detection and image semantic segmentation for applications such as autonomous driving [44] [97], biomedical images analysis [43] [184] and edges extraction [102] [221] [228] require solutions to achieve high precision at the pixel level. Pixel-level CNN methods have been experimented with success in biomedical images analysis. Cireşan et al. [43] proposed a binary pixel-based CNN for the detection of mitosis in breast cancer histology images. The network is trained with patches extracted from the images, classified as mitosis or non-mitosis based on the probability of the center pixel of being close to the centroid of a mitosis.

Figure 2.2: *Sample image from GeoPose3k data set [25] with original image (left) and render generated from DEM for the corresponding location (right).*

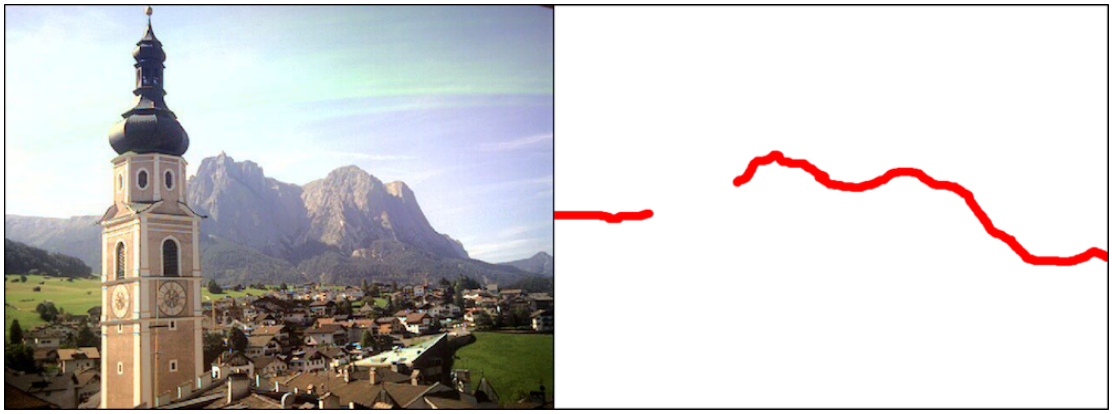


Figure 2.3: *Sample image from our manually annotated data set used for skyline detection considering obstacles.*

Pixel-wise CNNs are also used for edges extraction problems. In [221] the author took image patches as input and predicted whether their central pixels belonged to an edge. Furthermore, the same approach has also been applied in [2], where small patches of 16x16 were extracted to predict whether they contained part of the skyline or not.

Our skyline detection approach works at pixel-level. Similar to [2], we consider an image as a map of patches and analyze the local context around each center pixel to predict whether it belongs to the skyline or not. Differently from [43] and [221] we specialize the CNN for mountain skyline detection; differently from [2], [43] and [221], we use a Fully Convolutional Network (FCN) [140], which permits us to feed as input and process entire images fast, instead of extracting, processing and building a classification map for the individual patches; differently from [2], [3] and [169], we tackle the presence of occlusions interrupting the skyline, we train the network on a large data set of images ($\approx$ 9,000) taken in uncontrolled conditions including samples with many different types of obstacles, and we evaluate the obtained precision quantitatively taking into account such occlusions. We have also trained another CNN that works at column-level and is able to complement our pixel-level CNN boosting the accurate detection of obstacles and is suitable to apply as a second step to all the previous works not considering obstacles. Unlike all the previously mentioned works, we target the

fast execution of the CNN on both server and low power devices, in real-time and at the client.

Finally, it is worth mentioning that FCNs, as the ones used for this work, have recently evolved into models that rely not only on local features based on patches, but also on global features by training on entire images end-to-end [37] [232]. Such models are usually intended for problems such as semantic or instance segmentation, which require high quality annotations particularly hard to collect [44] [97] and tend to demand more computational resources. This work relies on annotations in the form displayed in Figure 2.3, which have been preferred in order to minimize the annotation effort, while maximizing the potential model accuracy and efficiency w.r.t. the different solutions explored and discussed in this Section.

2.2 Augmented Reality

Augmented Reality (AR) [10] is a well-established research area within the Human-Computer Interaction field [181], in which the users are offered an interface that enriches their view of the real-world with computer-generated information. The survey in [21] recaps the history of research and development in AR, introduces the essential definitions at the base of the discipline, and positions it among other related technologies. The authors also propose design guidelines and examples of successful AR applications, and highlight directions of future research.

AR systems are normally implemented on portable devices and have recently gained momentum due to the introduction by major hardware vendors of consumer-grade AR wearable devices (e.g. Microsoft HoloLens, Magic Leap One and Google glasses). Furthermore, a recent trend shows mobile devices being used as low cost AR platforms without requiring ad-hoc hardware [110]. Example applications are found in diverse areas, such as tourism [120], manufacturing [147], retail [47], construction [108], medicine [30], education [14], cultural heritage [211], games [161], etc.

2.2.1 Mobile Augmented Reality

Mobile Augmented Reality (MAR) [110] relies on the introduction of AR within mobile devices, which are highly portable, lightweight and already part of most people's daily lives due to their worldwide massive and increasing adoption [162]. It benefited from the improved standardization, increased computational power capabilities, and availability and integration of heterogeneous sensors.

An important branch of the discipline is the development of outdoor MAR apps to navigate [214], identify [48] and track [180] points of interest in urban or rural scenarios [120] [176]. Outdoor Augmented Reality applications exploit the position and orientation sensors of mobile devices to estimate the location of the user and her field of view, so as to overlay such view with information pertinent to the user's inferred interest. These solutions are finding promising applications in diverse sectors, where they replace traditional map-based interfaces with a more sophisticated user experience, whereby the user automatically receives information based on what they are looking at, without the need of manual search. Examples

of such AR apps include, e.g, Metro AR and Lonely Planet's Compass Guides[1]. The main challenge of such applications is to provide an accurate estimation of the user's current interest and activity, adapted in real-time to the changing view.

Commercial AR Software Development Kits (SDKs) power the rapid construction of AR applications on multiple platforms (e.g. smartphones, wearable devices, computers and web-based visualizers) by providing general-purpose components for recognition, tracking and rendering of augmented content. Such content can be visualized in either 2D or 3D, and even include animations. The work in [6] presents a complete comparison between multiple AR SDKs, such as Vuforia[2], Wikitude[3] and ARmedia[4]. However, traditional SDKs generally rely on marker-based detection, GPS and orientation sensors or on the specific a priori known appearance of certain objects, without actually considering the sensor readings' noise, the non-stationary nature of outdoor environments, and the value and complexity of the visual content captured by the camera of the device. These limitations prevent the possibility for the AR application to precisely overlay augmented content onto the view, provide high quality outdoor experience, and index visual content for supporting search, retrieval and extraction of semantic information of the annotated visual objects. Examples are sky maps, which show the names of constellations, planets and stars based on the GPS position and compass signal. An obvious constraint of this approach is that they may provide information that does not match what the user is seeing well, due to errors in the position and orientation estimation or to the presence of objects partially occluding the view. We present a novel framework customized for the development and fusion of marker-less location, orientation and visual information on MAR systems to refine the compass-based AR performance without knowing the appearance of the objects a priori and exploiting DL on-board to achieve such aim.

Recently, Apple released ARkit[5] for the development of AR apps for iOS devices (with A9 chip and above) and Google has followed the same direction with ARCore[6] for Android (supported for a set of devices[7]). Both SDKs have been evolving to introduce more features to understand the user's surrounding environment (e.g. depth, objects size and light estimation). Moreover, several commercial applications have achieved significant popularity in the last few years by means of successful immersive AR solutions to attract users' attention [197] and such market is projected to become huge within the next few years [138]. Pokemon Go[8] is a first class of outdoor AR mobile application developed by Niantic, which revolutionized the gaming scene by beating all-time records impressively fast in terms of downloads, active users and gross revenue. Besides, the use of advanced Computer Vision techniques has also contributed towards the development of better AR products and entertaining user-experience. Face detection and pose estimation [174] for the identification of keypoints have been exploited with high success

[1] http://www.lonelyplanet.com/guides
[2] https://developer.vuforia.com
[3] https://www.wikitude.com
[4] https://dev.inglobetechnologies.com
[5] https://developer.apple.com/augmented-reality/arkit
[6] https://developers.google.com/ar/
[7] https://developers.google.com/ar/discover/supported-devices
[8] https://www.pokemongo.com

by Snapchat for the application of face filters and animations, and was subsequently followed by Facebook, Instagram, Apple, etc. The retail and marketing sector has conceived multiple MAR innovative solutions, which work in real-time and include: ModiFace[9] for the virtual application of beauty makeup on the user's face, Wanna Kicks[10] to try-on sneakers with AR, Snapchat and Facebook AR animations to advertise brands (e.g. try-on Michael Kors' sunglasses with AR), and Ikea Place[11] to allow users to virtually place furniture in their homes.

2.2.2 Mountain exploration applications

A prominent class of outdoor MAR applications, particularly meaningful for this thesis, has been published for mountain tourism [115] [137]. Nowadays, numerous mobile applications have the goal of identifying and visualizing relevant information regarding mountain peaks (e.g. name, distance and altitude) and meta-data for other related items (e.g. shelters, trekking paths, etc) on top of the camera screen, thus attracting millions of people devoted to mountain activities around the world. The augmented view can be used on-line to enable learning and exploration, and also off-line, e.g. by saving annotated images in personal albums shareable within one's community. A data collection task could be easily embedded within the interface of a MAR application, e.g. by engaging users to take augmented photos during their outdoor activities and saving such enriched pictures and accompanying meta-data in a central repository, where they can be exploited in the study of mountain-related processes [32]. Considering the fact that these applications are meant to be used in mountain areas where Internet data access is not always granted, an off-line maps download management is an essential feature.

Some of the best-known publicly available apps are PeakFinder[12], PeakAR[13], ViewRanger[14], PeakVisor[15] and PeakLens[16]. PeakFinder uses the GPS position to compute from the DEM and visualize a virtual panorama with the mountain peaks positioned on the screen. It also exploits the compass and the user's orientation, but does not analyze the real images and has recently added the overlay of information on the camera view. Regarding the management of off-line maps, it automatically downloads the required area when the application is started based on the current position (e.g. the entire Alps). PeakAR uses the camera of the device, as well as other sensors, for the projection of the peaks on the screen. However, its implementation does not perform any geometrical projection of the terrain before positioning the peaks on the screen, which means that even peaks that are masked by the terrain configuration, and thus are invisible, are still shown. The problem is somehow alleviated by a simple filter that disables the display of peaks that are beyond a distance threshold. PeakAR manages off-line areas by default, because it requires downloading all the peaks around the world

[9] http://modiface.com
[10] https://wanna.by
[11] https://www.ikea.com/gb/en/customer-service/mobile-apps
[12] http://www.peakfinder.org/mobile
[13] https://www.salzburgresearch.at/projekt/peakar
[14] http://www.viewranger.com/skyline
[15] http://peakvisor.com
[16] http://peaklens.com

beforehand. This is possible because only the 3D coordinates of the peaks are downloaded, and not the surrounding DEM points. Several other apps, similar to the above mentioned ones, use only the position and orientation sensors, which are imprecise and may induce substantial peak positioning errors. Some clear examples are ViewRanger and PeakVisor. ViewRanger targets trekkers and offers route guides and GPS navigation; recently it incorporated an AR function, which overlays points of interest such as peaks, towns, lakes, cliffs and glaciers, over the camera view. Positioning uses only the GPS and orientation sensors. PeakVisor allows the user to correct compass errors manually, by registering the virtual panorama with the real image captured by camera, using the sun position as a hint. Again, sensor errors may intervene: e.g. the DEM resolution is such that the virtual panorama generated from it does not always match the camera image well, which makes manual adjustment hard. Porzi et al. [168] proposed an app for mountain peak detection, but said work is not publicly available, does not focus on the time efficient peak identification and does not address mobile AR requirements, such as real-time response, asynchronous dynamics of the algorithms and uncertain internet connection.

As a use case and part of this work, we have developed PeakLens app, which is reviewed in Section 2.6.4. In contrast with all the previously mentioned apps, it aims at providing high precision peak identification and information overlay by using Artificial Intelligence to analyze the frames captured by the device camera and accurately position the points of interest. Its core is the comparison of what the user sees with the 3D model of the terrain and the automatic alignment of the virtual and real mountain skylines, whereby the app can correct significant sensor errors. Moreover, PeakLens can also handle occlusions, in order to avoid displaying peaks when they are masked by an object in front of them (e.g. a bell tower, a person, or a tree).

2.3 Artificial Intelligence on the edge

Artificial Intelligence (AI) [188] is a vast field that studies the creation of computer systems capable of mimicking the human cognitive functions in order to solve non trivial problems. It comprises several related concepts, and a visual hierarchical overview is provided in Figure 2.4.

Machine Learning (ML) [22] is a subfield of AI that develops solutions that do not rely on explicitly programmed instructions to perform a certain task, but exploit a data-driven approach in which patterns are learned from training data.

Artificial Neural Networks (ANN) [23] [83] are ML computing systems composed of interconnected group of processing elements (also called artificial neurons) arranged in a series of layers, which are inspired by the biological visual cortex structure. Based on the number of hidden layers, i.e. the layers between the input and output layers, they can be categorized into two groups: shallow, when they contain only one hidden layer, and deep, when they contain multiple hidden layers.

Deep Learning (DL) [129] is a class of ML algorithm that relies on Deep Artificial Neural Networks to learn complex data representations through mathematical

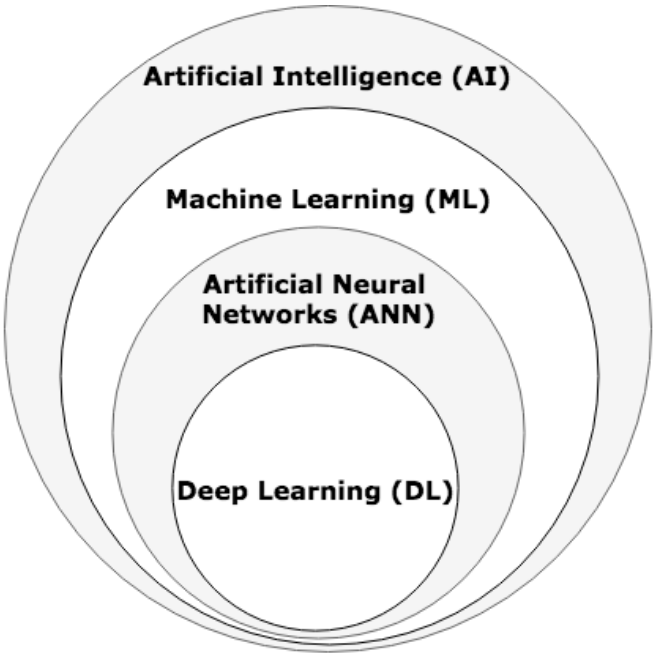


Figure 2.4: *Overview of AI-related concepts hierarchy.*

processes, such as backpropagation [186]. DL has recently experienced great success, thanks to algorithmic improvements [68] [107] [156] [195] [205] and the vast amount of training data and the increasing computational power available nowadays [39]. In particular, DL models based on feed-forward Deep Convolutional Neural Networks (CNN) [121] have proved capable of achieving high quality results in a wide range of Computer Vision tasks, such as image classification, detection, localization and segmentation [73]. CNNs can solve a target problem by extracting complex non-linear feature hierarchies from the input space of the training data.

Typical implementations of DL models focus on the maximization of accuracy for a given task, and architectures to achieve such an objective have become significantly deeper and more complex over time [86] [98] [202] [209]. Powerful workstations with Graphics Processing Units (GPUs) were fundamental for the success of DL, making its computationally expensive training possible. On the other hand, even though resources of embedded systems, such as smartphones, tablets, wearable devices, drones and Field Programmable Gate Arrays (FPGAs), are rapidly improving, they are still not completely suitable for the deployment of big and complex models [123] [143]. Furthermore, the use of remote cloud services for the execution of models has its own drawbacks related to the use of the network, such as cost, coverage, availability, latency, security and privacy issues [55]. All the above-mentioned limitations promote the interest in alternatives expressly conceived for efficient deployment on the edge [124] [127] [219], which are covered in this Section.

2.3.1 Compression techniques

Compression techniques [40] target large scale, redundant or computationally demanding architectures and aim at reducing the model size, number of parameters and floating point operations (FLOPs), possibly tolerating small accuracy drops in favor of execution acceleration and optimization of computational resources, storage, memory occupation and energy consumption.

Quantization [71] [99] [227] [233] reduces numerical precision to represent CNN associated weights (e.g. from 32 bits to 8 or 16 bits), so as to accelerate run-time performance and reduce storage and memory overhead, with minor or at least not massive accuracy loss. Wu et al. [227] proposed CNNs quantization techniques for mobile devices and applied such approach over state-of-the-art models achieving 4 $\approx$ 6x speed-up and 15 $\approx$ 20x compression with merely 1% loss of accuracy. Additional techniques applicable during the model training rely on low-precision fixed-point operations [74] [103] and more intensive approaches even consider weight representations binarization [45] [175]. Furthermore, post-training quantization can be further improved as demonstrated by Finkelstein et al. [63].

Pruning [80] [88] [133] removes redundant connections, thus reducing the number of weights, and proved to efficiently compress state-of-the-art models by one order of magnitude. Early approaches were also applied with the objective of reducing over-fitting [82] [131]. Han et al. [79] presented a three stage pipeline, also based on pruning, to which quantization and Huffman coding were added for further reducing disk storage without compromising accuracy. Nonetheless, conventional compression techniques may be sub-optimal, time consuming, and require relevant domain expertise that limits their application. He et al. [87] proposed the use of AutoML [236] for Model Compression (AMC), an approach to handle the model compression and acceleration on mobile devices by means of reinforcement learning, thus achieving more optimal results in an automated manner.

Alternative techniques include knowledge-distillation [11] [28] [92] to compress and transfer knowledge from complex models to simpler ones, matrix sparcification [19] and tensor decomposition methods [118] followed by low-rank approximation [53] [109], for the reduction and compression of weights.

The effectiveness of compression depends on the size and redundancy of the original model and most compression techniques are applicable either after or at training-time. In general, post-training compression is easy to apply, but may induce a sensible accuracy loss, especially when no fine-tuning is performed on the models afterwards. On the other hand, training-aware compression tends to achieve better results, but requires more time and it is more complex to perform.

2.3.2 Optimized model architectures

Lightweight architectures with compact layers pursue the design of an optimized network topology, yielding small, fast and accurate models, suitable for resource-constrained devices.

SqueezeNet [105] is a first-generation optimized CNN architecture based on

fire modules with small Convolutional kernels; such modules are composed of squeeze layers (Pointwise Convolutions, , i.e. 1x1 convolutional filters applied to the total number of channels of the activation map), that reduce the number of input channels, thus parameters and computation, and expand layers (concat of 1x1 and 3x3 Convolutional filters), that restore the input depth; downsampling is delayed within the network so as to achieve higher accuracy by exploiting larger activation maps and fully connected layers are replaced by global average pooling; it achieved the same accuracy as AlexNet [121] with 50 times less parameters and can be effectively compressed on disk [79] up to 510x w.r.t. AlexNet.

MobileNet [94] is a family of efficient models for mobile vision applications, which perform different trade-offs in terms of accuracy, computation and number of parameters. Such models, released by Google, are based on the introduction of Depthwise Separable Convolutions [42] and have outperformed most of the previous state-of-the-art models (e.g. MobileNet comprises same size, +4% in accuracy, and 22x less computational time w.r.t. SqueezeNet). Depthwise Separable Convolutions emerged as a highly computational efficient yet accurate building block that factors a standard Convolution into a Depthwise Convolution, i.e. convolution applied to each single channel at a time, followed by a Pointwise Convolution. Afterwards, MobileNet v2 [191] further improved MobileNet v1 by incorporating the inverted residual with linear bottleneck module, which uses identity residual shortcuts [86] to connect low-dimensional information bottleneck tensors.

Also DenseNet [95] exploits a variation of the residual block by introducing direct connections, not just between consecutive layers, but between any two layers with the same feature map size in the network, consequently achieving state-of-the-art performances while requiring fewer parameters, layers and computation.

Several other efficient approaches have been proposed, including ShuffleNet [144] [231] with its low-cost Group Convolution and Channel Shuffle operations to reduce the Mult-Adds; DeepRebirth [132] for slimming consecutive and parallel non-tensor and tensor layers; adaptive deep learning model selection [212] based on the input to improve accuracy and reduce inference time for embedded systems; Fast-SCNN [171] for fast semantic segmentation with multiple resolution branches simultaneously capturing and combining high and low level resolution features.

Recently, reinforcement learning and automated approaches have also been exploited for the discovery of highly accurate efficient building blocks by using Neural Architecture Search (NAS) [15] [236] to support and alleviate the burden of manual design and hyperparameters tuning. Tan et al. [210] proposed MnasNet, an automated hardware-aware NAS that exploits a multi-objective reward to address both accuracy and latency measured in real-world mobile devices. Wu et al. [226], further improved hardware-aware NAS methods by proposing an efficient differentiable NAS version with gradient-based optimization, which is able to achieve high performance in terms of both accuracy and latency for specific target devices, while also significantly reducing the computational needs for its execution.

2.3.3 Hardware acceleration

Hardware acceleration (HA) is the use of dedicated hardware to complement general-purpose CPUs and perform computationally intensive work more efficiently, e.g. by favoring specific operations and data-parallel computation. This includes the use of heterogeneous processors, such as Digital Signal Processors (DSPs) for energy optimization [126], GPUs for computation optimization [101] and, more recently, Neural Processing Units (NPUs), as prominent mobile system on chip (SoC) vendors have incorporated specialized hardware for accelerated AI inference, focusing on vector and matrix-based instructions. Nonetheless, such instructions and the access to them are typically unavailable or depend on the proprietary primitives and Software Development Kits (SDKs) of each specific vendor, which are incompatible and impair the porting of acceleration solutions. Qualcomm Snapdragon Neural Processing Engine (SNPE) [17] (SNPE) and Arm NN[18] are clear examples of it. Given the need of standardization, Google has recently published the Android Neural Networks API[19] (NNAPI), which defines a layer of abstraction that provides unified access to DL run-time acceleration. Its support for current devices is still limited due to its availability from Android 8.1 and requires specialized vendor drivers, otherwise computation falls back to the CPU. Similarly, recent versions of OpenGL[20] and Vulkan[21] introduced compute shaders for GPU-based efficient non-graphic computations, shared memory and intra-group synchronization, but their support is reduced for older devices and depends on vendors' implementation. From iOS 8, Apple devices feature the Metal API[22], designed to maximize performance and let developers access HA. Apple has the advantage of targeting a limited and relatively homogeneous set of devices, while having full control over the production, which simplifies integration and support.

Several studies [38] [75] [77] [78] [182] have pursued the further optimization of specific target models inference by means of designing highly customized ad-hoc hardware architectures for inherent needs, such as efficient sparse matrix-vector multiplication in the case of pruned models.

The introduction of efficient DL building blocks has been studied along our work in order to enhance the suitability and deployment of DL models on devices with constrained resources.

2.3.4 Heterogeneous computing scheduling

While HA relies on dedicated physical components designed to speed-up specific operations, heterogeneous computing scheduling comprises the design of strategies to efficiently coordinate and distribute the workload among processors of different types [4]. Previous research works [101] [122] have proposed DL scheduling techniques for embedded systems. Results show a good level of optimization, with accuracy loss up to 5%. However, for maximum efficiency, these methods require

[17] http://developer.qualcomm.com/software/qualcomm-neural-processing-sdk
[18] https://www.arm.com/products/silicon-ip-cpu/machine-learning/arm-nn
[19] https://developer.android.com/ndk/guides/neuralnetworks
[20] https://www.opengl.org
[21] https://www.khronos.org/vulkan
[22] https://developer.apple.com/metal/

specific drivers (e.g. to support recent versions of OpenCL) or custom implementations for different architectures with direct access to hardware primitives.

2.3.5 Mobile DL Frameworks

Frameworks for the execution of DL models on mobile and embedded systems pursue optimized deployment on devices with limited resources, by managing memory allocation efficiently, to avoid overloading, and exploiting the available hardware resources at best for acceleration (e.g. the GPU). We built PolimiDL, our own optimized framework for DL acceleration on mobile devices and embedded systems, when no efficient off-the-shelf solutions were available; recently, some new tools to support the execution of machine learning on mobile devices were released, e.g. TensorFlow Lite[23], Caffe2[24], Paddle[25] and Core ML[26]. Training is performed off-board, with mainstream tools such as TensorFlow, PyTorch, Caffe or MXNet, and the resulting models are converted into the format of the mobile framework for deployment. Open Neural Network Exchange Format[27] (ONNX) proposes the standardization of models definition, to simplify the porting of models trained with different tools. Furthermore, CoreML already exploits Metal HA on iOS devices, while NNAPI support for Android frameworks and devices is still neither totally stable nor fully integrated.

2.3.6 Benchmarking

Performance benchmarking measures indicators to compare run-time architectures. For mobile DL, relevant metrics include accuracy, execution time, memory overhead, and energy consumption. Shi et al. [200] assessed the performance of various open-source DL frameworks, by executing different models over a set of workstations with heterogeneous CPU and GPU hardware. Pena et al. [163] measured inference time and power consumption of TensorFlow and Caffe over a set of embedded systems. The work in [198] defined guidelines to assesss DL models on Android and iOS devices, and [81] studied the latency-throughput trade-offs with CNNs for edge Computer Vision. Finally, Ignatov et al. [106] presented a complete overview regarding the state of Deep Learning in the Android ecosystem and created a publicly available mobile app (AI Benchmark) to benchmark performance on a set of DL Computer Vision tasks. Scores are calculated by averaging performance results over all the user devices and the corresponding SoCs evaluated.

We benchmarked our optimized DL framework and obtained highly competitive results w.r.t. TensorFlow Lite for the execution of small DL models on mobile devices.

[23] http://www.tensorflow.org/mobile/tflite
[24] http://caffe2.ai/docs/mobile-integration.html
[25] http://www.paddlepaddle.org/docs/develop/mobile
[26] http://developer.apple.com/documentation/coreml
[27] https://onnx.ai

2.4 Multi-sensor applications testing

Developing and testing applications that operate in complex working conditions has become a prominent research task, fueled by the widespread adoption of mobile applications that employ multiple sensors [111] [153] [206].

In the software engineering literature, the general conditions in which an application operates are abstracted into the concept of *context* [1], [18], defined as the information that characterizes any entity relevant to the interaction between the user and an application. Context-aware development has been specifically studied in the case of mobile applications [36], which provide a particularly rich notion of context that embraces the user's state, the device capability, the sensed environment, and the network connectivity state.

Testing context-aware applications is a special sub-topic of context-aware software development, which recasts the classical methods of conventional application testing to the specific case in which the system under test requires the supply of context information.

The recent work [179] focuses on the generation of context information for the purpose of testing mobile applications in an emulated environment. The authors model the context as a set of *modalities*, each of which corresponds to a facet of the contextual information, such as network connectivity, position, motion sensors, and camera. They illustrate the design of a tool, called ContextMonkey, which fetches data for each context modality from heterogeneous sources, builds an integrated context stream and feeds such stream to the emulation environment, where it is exploited for running a test session. ContextMonkey is evaluated primarily with respect to its capacity of supplying the context information to an application inside the emulator *with fidelity*, i.e., at the same rate as in the real working conditions. An interesting collateral finding of the assessment is that the synthetic, model-driven construction of multi-sensor context streams, evaluated in a mobility use case, could not fully reproduce the semantic complexity of the real context streams recorded in the field; this observation is one of the motivations of our capture-based approach. Our work shares with [179] the focus on multi-sensor application testing; however, differently from ContextMonkey, our focus is not the fidelity of the replay of context streams during emulation, but the use of multi-sensors usage traces recorded in the field for the discovery of soft errors.

The VanarSena tool [177] instruments the binary code of the application to perform testing in a way that achieves both coverage and speed. The tool runs on a cloud and lets developers upload the application binary code and run multiple test sessions in parallel to emulate user behavior, network conditions, and sensor data, returning a failure report.

The dynamic testing of (non multi-sensor) mobile applications via controlled execution has also been pursued in a number of works. For example, Machiry et al. [145] describe a system, called Dynadroid, whereby developers can observe, select, and execute Graphical User Interface (GUI) and system events in a mobile device emulator, so as to drive black box analysis of test runs. Other related studies mostly focused on capture and replay at the level of GUI input-output events, without considering the specificity of mobile devices [57] [224]. Conversely,

Gomez et al. [70] present an approach specifically conceived for mobile devices, in which they record and replay Android apps usage traces by replicating GUI gestures and sensor readings. However, their tool cannot replay certain services such as camera preview and GPS location, which are critical signals for sensor- and location-based applications. Our approach is similarly based on the observation of application runs, but focuses on capturing and replaying multi-sensor data; it could be extended with a system and GUI event capture, as in [70] and [145], to create test sessions that span all categories of input events: sensor, UI and system.

The use of a capture and replay approach for testing of mobile applications is reported in [114]; the authors present a tool for the dynamic analysis of executions, the debugging of deployed applications, and regression testing. A relevant finding is that the effectiveness of regression testing highly depends on how well the tool reproduces the way the program is used in the field. The accomplishment of such an objective in a multi-sensor mobile application requires a non trivial capture and replay architecture, which is a main contribution of our work.

2.5 Crowdsourcing and Citizen Science

Crowdsourcing [58] is a collaborative model based on the outsourcing of specific tasks to a group of participants committed towards the solution of a common cumulative goal, which requires the collection, validation and/or processing of data, usually via internet-based systems.

Citizen science [149] refers to the direct engagement of not necessarily specialized individuals (the citizens) to help address scientific problems by collaborating through crowdsourcing systems.

The massive diffusion of social media, with its powerful tools for public communication, engagement, and content sharing, has multiplied the ways to engage volunteers and exploit relevant public User-Generated Content (UGC). In particular, social media combined with mobile devices favored the collection of *geo-located* UGC in applications related to spatial information, so-called Volunteered Geographical Information Systems (VGIS), in which citizens help enhance, update or complement existing geo-spatial databases [72]. OpenStreetMap[28] (OSM) [76] is a clear example of a well-known, large scale collaborative project for the crowdsourced collection of geographical information (e.g. roads, landmarks and multiple points of interest) worldwide. It relies on a VGIS and counts on an active community of millions of mapping contributors distributed all around the world. OSM data is open and represents a rich source of information that can be exploited for several geolocation-based applications [89] [172] [194]. Furthermore, OSM data has also been used in our research work.

2.5.1 Platforms

Platforms such as Amazon Mechanical Turk[29] (MTurk) [29] and Zooniverse[30] [203] are exploited in diverse scientific projects that require the collection and processing

[28] https://www.openstreetmap.org
[29] https://www.mturk.com
[30] https://www.zooniverse.org

of data. Moreover, with the progress of Deep Learning techniques, the annotation of vast data collections for training Neural Network models has become an essential task for many research projects in diverse fields of application. In such a context, crowdsourcing arises as a suitable sourcing approach to obtain the necessary labeled data [52]. MTurk relies on paid workers and is widely used for the collection of data for academic, open source, and private commercial purposes. Conversely, Zooniverse relies on citizen scientists, i.e. domain experts and passionate volunteers motivated by the sole idea of helping to solve a problem of common interest. SciStarter[31] [93], a research affiliate of Arizona State University, is an online platform that exposes a searchable database for citizen science projects. Such platform pursues the preservation of participants' interest over an extended period of time and focuses on user engagement by providing participants with tools to track contributions and time devoted to a project, to contribute in multiple projects across different disciplines, and to have collaborative relationships with project owners and other participants; it also exploits user location to promote locally relevant projects.

Crowdsourcing solutions represent a great opportunity because of their ability to collect or generate large and varied volumes of data. However, such data are not always used to their full potential due to concerns regarding quality [5], transparency of the collection process [167], or because the data sets are not released publicly [158]. Several works address the problem of crowdsourced data quality and propose techniques to identify and improve noisy labels [199] [204] [225]. CitSci.org[32] [222] is a platform created by the Natural Resources Ecology Lab (NREL) at Colorado State University, which aims at addressing these problems by providing project coordinators with tools to document research goals, data collection protocols and methodologies, data sample selection criteria and the data quality procedures, to improve data reuse across related research efforts; it also supports the specification of metadata, both common and specific of different types of research programs, to help contextualize the data and improve their reuse [141]. Finally, CitSci.org encourages the publication and the exchange of data sets in standardized formats through web services, so as to establish scientific authority and improve credibility [158].

2.5.2 Environmental monitoring

Environment data collection increasingly exploits the contribution of citizens, who cooperate with the acquisition and processing of large geo-referenced data sets to extract usable information from them, so as to exploit such knowledge in the study of natural and anthropic processes [20] [49] [50] [113] [116] [142] [148] [151] [166].

Several approaches have been applied to disaster management for e.g. earthquake mapping [235] and rapid flood damage estimation [170]. Applications monitoring hazards through the collection of user-generated content are also reported: tweet distribution analysis for monitoring is employed in [189] for earthquakes and in [196] for floods. Examples exist of continuous monitoring applications in the environmental field: bird observation network [207], phenological studies [178],

[31] http://scistarter.com
[32] http://www.citsci.org

hydrological risk assessment [51], plant leaf status assessment [157] and geological surveys[33]. Besides text, also visual content, such as Flickr photographs [220] and public touristic webcams [155] have been used to monitor environmental phenomena, such as coarse-grained snow cover maps [220], vegetation cover maps [230], flora distribution [220], pollination conservation [128], cloud maps [155] and other meteorological processes [104].

There are two main mechanisms that can be applied for large scale environment geo-data collection. One is crawling geo-located images from multiple web data sources at scale, which requires the definition of a processing pipeline to automatically retain only data of interest. The other option relies on the implementation of ad-hoc crowdsourcing systems, which are more robust and ensure the information completeness and user validation.

The diffusion of mobile phones and of mobile applications linked to social networks and to content sharing sites boosted a wave of geo-referenced data collection applications, which exploit the fact that people carry during their outdoor activity an Internet-connected device equipped with a variety of sensors, including camera, microphone and GPS. This phenomenon spawned the release of mobile applications for earth observation and environmental monitoring [62] and also of frameworks that let non-programmers build mobile data collection tools for citizen science campaigns [117]. Project Budburst[34] [178] gathers information regarding the flowering of native plants for climate change studies, engaging volunteers to upload timestamped, geo-tagged plant photographs. Other examples of the use of mobile phones for crowdsourcing environmental and ecological data include: avian surveys [216], water level monitoring [142], noise pollution [146], seismic early detection [119], mosquito surveillance [154], biodiversity observation [217], botanic monitoring [69] and meteorology monitoring [116].

2.6 Use case

Environmental data processing through the analysis of low-cost, large scale, geo-located multimedia data can be exploited for the enhancement of diverse monitoring scenarios. However, one major challenge for the development of environment crowdsourcing solutions consists of offering citizens a useful, satisfying and possibly entertaining experience, so as to motivate them to participate, use the applications on a frequent basis and spread the word about it to their social circles [69] [207].

As a use case, this work included the implementation of PeakLens, an outdoor AR mobile app that identifies mountain peaks and overlays them in real-time on the view; PeakLens processes camera frames in real-time by using a CV-powered module for enhanced positioning of augmented markers. The application, besides providing a nice experience to the user, can be extended and employed to crowdsource the collection of annotated mountain images for environmental monitoring applications [32]. Previous works on mountain analysis [32] and SnowWatch project [61] were fundamental precursors of this research and are covered in the following subsections.

[33] http://britishgeologicalsurvey.crowdmap.com
[34] https://budburst.org

2.6.1 Mountain analysis

Image analysis in mountain regions is a well investigated area, with applications that support environmental studies on climate change and tourism [56]. As reviewed in Section 2.1, the identification of natural landscapes and mountains in public photographs taken in uncontrolled conditions represents a challenge because vegetation, illumination and seasons affect appearance and visibility [12]. Besides, a prominent application field of mountain image analysis is snow information extraction to address the problem of water availability in mountain regions, where the water supply is mostly conditioned by the snow coverage [160]. Traditionally, snow is monitored through manual measurement campaigns, permanent measurement stations, satellite photography, and terrestrial photography. Several approaches [185] [190] rely on cameras designed and positioned ad-hoc by researchers, to segment the portion of the photograph corresponding to a certain mountain in snow covered areas. Zhang et al. [230] proposed the application of web media mining by analyzing tags and visual features of geo-tagged Flickr images, to predict vegetation and snow cover. Finally, Castelletti et al. [32] proved that user generated mountain pictures and publicly available touristic webcams can be analyzed to compute snow indexes, usable for improving predictive water systems operation [32].

2.6.2 SnowWatch

SnowWatch project[35] [61] tackles the problem of monitoring mountains for the collection of public Alpine images and the extraction of snow indexes usable in water availability prediction models. To this aim, SnowWatch crawls a large number of images from content sharing sites and touristic webcams, classifies those images that portrait mountain peaks and contain the location of shooting, identifies visible peaks by automatically aligning each image to a synthetic rendition computed from a public DEM, finds the pixels of each peak that represent snow and calculates useful snow indexes, such as minimum snow altitude and Snow Water Equivalent (SWE). These indexes are then used to feed existing water prediction models and compared with other official sources of information.

SWE time series are usually estimated through a hybrid of satellite retrieved information and ground observations. Ground stations and satellite data, however, have limits when used to investigate snow processes, which exhibit high spatio-temporal variability [230]. Ground stations are few and coarsely spaced, especially in high altitude regions. Satellite snow products have limitations in alpine contexts [54]: space-board passive microwave radiometers (e.g. AMSR-E) penetrate clouds and provide accurate snow cover estimation, but have coarse spatial resolution (25 km); active microwave systems (e.g. RADARSAT) detect the presence of liquid water content, but require additional ground observations to make accurate estimates, whereas optical sensors (e.g. MODIS) generate high spatial and temporal resolution maps, yet cannot penetrate clouds. To complement satellite and ground stations observations, SnowWatch project tested the use of public web cam images as a reliable source of snow information. More

[35] http://snowwatch.polimi.it/?lang=en

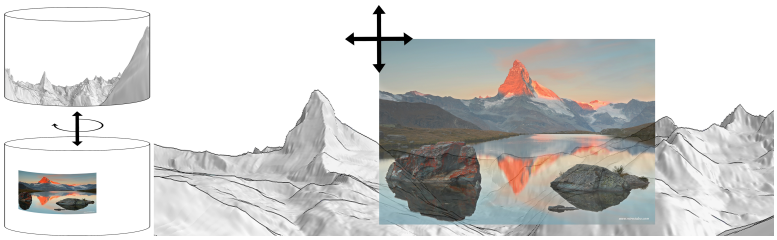


Figure 2.5: *Photo to panorama cylindrical and equivalent 2D Cartesian alignment.*

than 3,500 candidate web cams in the Alpine area from touristic, meteorological, and skiing web cam directories were identified and manually checked, to remove those that were not framing significant mountain slopes. Nearly 2,000 web cams passed the test and are continuously queried by a web crawler that checks each web cam at $1'$ frequency and processes all new images, acquiring from 10 to 1,500 images per day for each camera, depending on its update frequency and working hours. A bad weather filter is applied to the crawled images to discard those not suitable for further processing (a manual screening of 1,000 images from 4 web cams revealed that 33% of the acquired images on average are exploitable for analysis). The filtered images are then processed by a pipeline of components that geo-reference the peaks present in the image,normalize the daily shots into a daily median image, and classify the image pixels as snow or no-snow, producing a *snow mask* per image. Finally, time series of *Virtual Snow Indexes (VSIs)* are computed from the geo-referenced and time-stamped snow masks, which are used as a proxy of the snow covered area.

2.6.3 Offline Peak Detection for the Web

One of the key algorithms of SnowWatch Web architecture is the offline peak identification. Peak positions are obtained through the alignment between the photo and the terrain model. Given a photograph and the meta-data extracted from its EXIF container (geo-tag, focal length, camera model and manufacturer), a matching is performed with a 360° panoramic view of the terrain synthesized from a public, Web-accessible DEM. The rendered panorama contains the mountain peak positions, so once a correct overlap is found, peak positions are projected from the panorama to the photo. The alignment can be seen as the search for the correct overlap between two cylinders (assuming the zero tilt of the photograph): one containing the 360° panorama and the other one containing the photo, suitably scaled. As Figure 2.5 shows, this is equivalent to looking for the offset between the photo and the unfolded 2D panorama that guarantees the best overlap.

The alignment method proceeds in four steps, described below and illustrated in Figure 2.6.

Preprocessing: The horizontal Field Of View (FOV) of the photograph is calculated from the focal length and the size of the camera sensor. Then, the photograph is rescaled considering that the width of the panorama corresponds to a FOV equal to 360°. After this step, the photo and the panorama have the same scale in degrees per pixel and thus matching can be performed without the need of scale invariant methods. Then, an edge extraction algorithm is applied to both

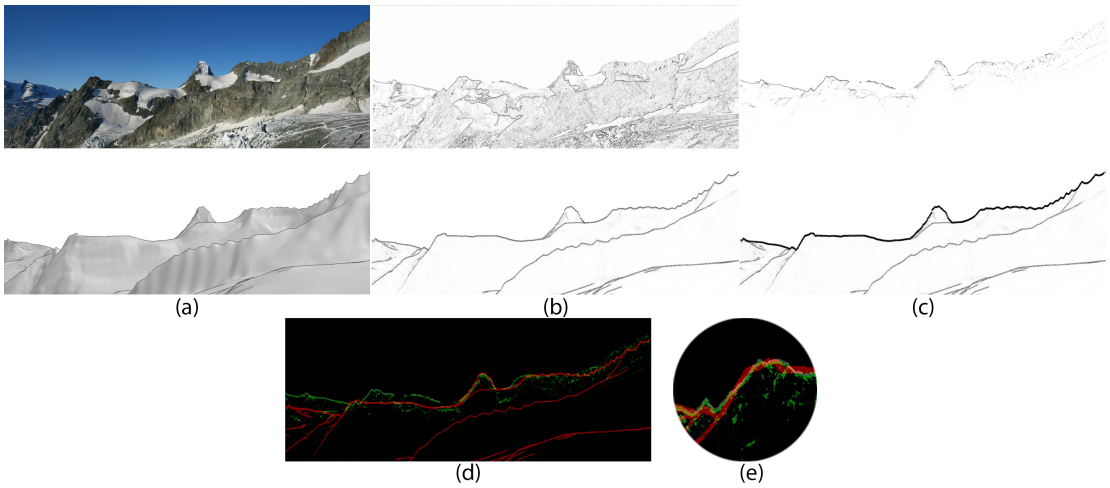


Figure 2.6: *An example of the photo-to-terrain alignment: (a) input photograph (top) and corresponding panorama (bottom), (b) edge extraction, (c) skyline detection, filtering and dilation (d) global alignment with refinement (e) local alignment.*

the photograph and the panorama to produce an edge map, which assigns to each pixel the strength of the edge at that point and its direction (Figure 2.6b).

Matching edges of an image with those of a virtual panorama requires addressing the fact that there is not a one-to-one mapping between edge pixels extracted from the two sources. The photo generates many noisy edges that do not correspond to the mountain slopes, but to other objects in the foreground (e.g. rocks, trees, lakes, houses, etc.) and in the background (e.g. clouds, snow patches, etc.). Thus, a skyline detection algorithm is employed [135], and all the edge pixels above the skyline are removed, being considered obstacles or clouds. Then, a simple weighing mechanism is applied, which assigns decreasing weights to the edge pixels as the distance from the skyline increases (Figure 2.6c - top). As for the panorama, the edges corresponding to the skyline can be simply identified as the upper envelope of the edge map, by keeping, for each column of pixels, the topmost edge point. Since the edge filtering of the photograph emphasizes the edges of the skyline, a morphological dilation is applied to emphasize the edges corresponding to the skyline of the panorama (Figure 2.6c - bottom).

Global alignment: The matching between the photograph and the corresponding panorama is performed using a Vector Cross Correlation (VCC) technique [13], which takes into account both the strength and the direction of the edge points. The output of the VCC is a correlation map that, for each possible horizontal and vertical displacement between the photograph and the panorama, indicates the strength of the matching.

Local alignment: to improve the precision of the position of each mountain peak, a local optimization is applied. For each peak we consider a local neighborhood centered in the photograph location identified as the peak position by the global alignment. In this way each peak position is refined by identifying the best match in its local neighborhood. Overall, this is equivalent to applying a non-rigid warping of the photograph with respect to the panorama.

2.6.4 PeakLens app

Mobile outdoor Augmented Reality applications are an emerging category of solutions that hold the promise to help design engaging user experiences suitable for geo-referenced data collection tasks. These applications are implemented in mobile terminals (mainly mobile phones, but also the forthcoming consumer-grade smart glasses) and enrich an outdoor experience, such as trekking or star gazing, by overlaying useful information onto the device camera view. Furthermore, the acquisition of images with crowdsourcing mobile applications can complement certain limitations comprised by public webcams. Web cams afford a good temporal frequency, but fall short in spatial coverage; they are positioned at fixed locations, often chosen for touristic purposes other than the selection of monitoring points of environmental interest. On the other hand, in the case of mobile applications, spatial coverage is larger and more uniform. Another advantage of crowdsourcing mobile applications is their dynamic nature: people can be engaged in data collection tasks at specific places and times.

PeakLens is a real-world Outdoor Mobile Augmented Reality app that aims at incorporating Artificial Intelligence to provide a high-quality and entertaining experience to users, while they can take photographs of mountain landscapes, in which all the visible peaks are precisely identified and geo-referenced. PeakLens is available for Android and has currently +500k installs worldwide. Figure 2.7 presents its user interface.

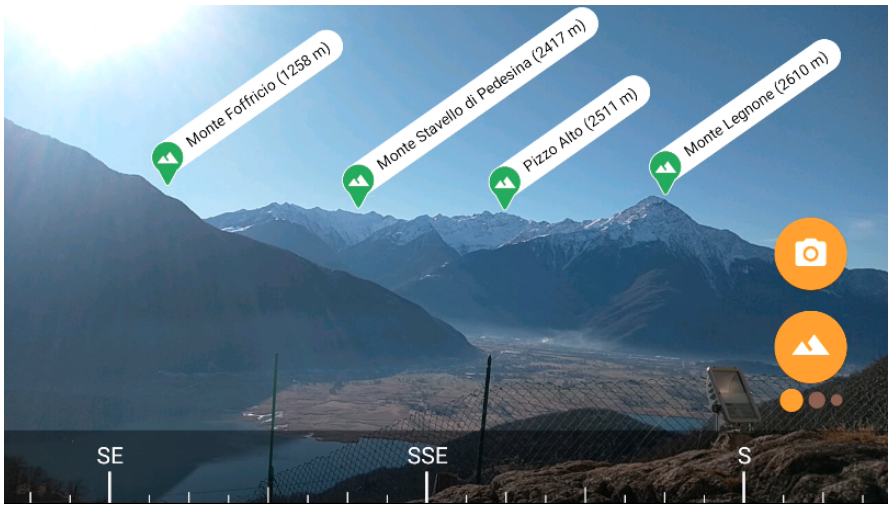


Figure 2.7: *PeakLens interface: compass orientation (bottom band); photo shooting command (upper button); peak scrolling (bottom button) and peak page indicator (3 circles).*

This AR application recognizes mountain peaks accurately in real-time and overlays their corresponding information (e.g. name, distance and altitude) on top of the view by using Artificial Intelligence and exploiting the visual context captured by the camera. Besides projections based on noisy mobile orientation sensors, it features a Computer Vision component based on Deep Learning (running on-board the phones with a proprietary framework) that analyzes the camera frame and detects the pixels in the image that correspond to the natural mountain skyline. Figure 2.8 presents an example of the app interface with the correspond-

ing detected skyline highlighted in red color. Such skyline is subsequently aligned with respect to a virtual terrain panorama based on the GPS location of the user. As a result, the final mountain peak projections are significantly improved, thus providing an enhanced user experience.

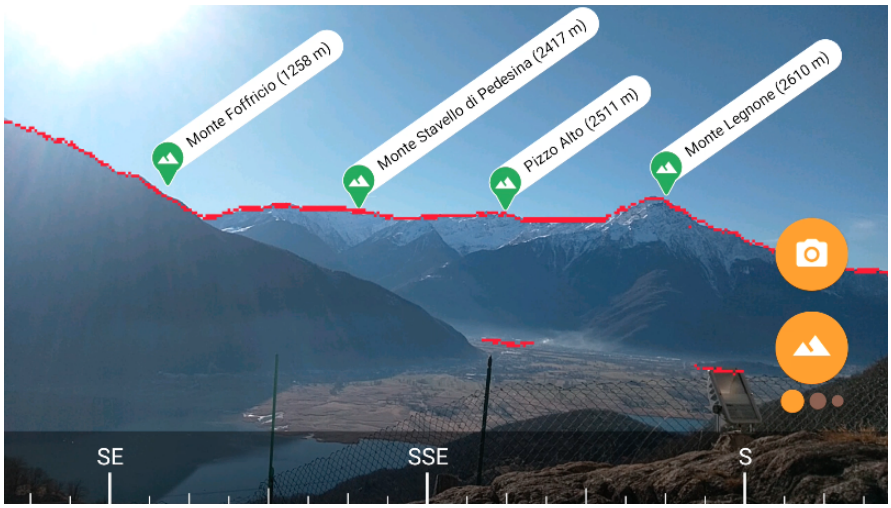


Figure 2.8: *The skyline extracted from the computer vision module from the frame of Figure 2.7.*

The peak identification and labeling function exploits a DEM of the Earth and a repository of peak metadata, and matches the mountain summits of the DEM to the skyline peaks extracted from the camera frame, to compute the correct 2D screen coordinates of the visible peaks. The matching procedure is the core of the application: the user's location, the device orientation values, and the camera field of view are exploited to generate a bi-dimensional virtual panorama from the DEM point cloud. Figure 2.9 shows an example of the virtual panorama generated from the DEM. Then, the DEM and the peaks metadata repository are queried to determine the list of visible peaks, given the position and orientation of the device; hidden peaks masked by the terrain configuration are excluded; the artificial skyline from the virtual panorama (shown in Figure 2.9) is aligned with the skyline extracted from the frame (shown in Figure 2.8) and the visible peaks are projected from the 3D space to the 2D space, obtaining the screen coordinates. Based on the 2D coordinates, the visible peaks are *ranked* by a visual relevance criterion, which is applied in the cases in which more peaks are visible than could be displayed on the device small screen. Finally, a GUI component selects the peaks to show based on the ranking and the size of the screen and overlays the peak positions and metadata, producing the visualization shown in Figure 2.7[36]. The motion sensors are used to trigger the re-computation of the 2D peak positions when the user moves the device. For offline usage, the DEM and the peak metadata repository have been segmented and compressed and can be downloaded and queried in the mobile device, in absence of Internet connectivity.

[36]The GUI comprises a *More peaks* button (bottom in Figure 2.7) to show the peaks that could not fit in the screen.

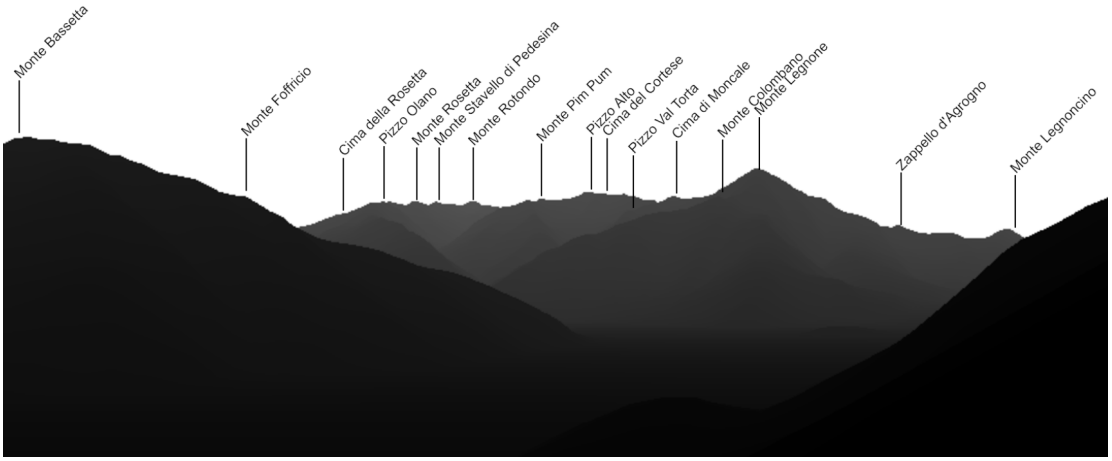


Figure 2.9: *The virtual panorama computed from the DEM, queried with same location and orientation of the device that produces the screen image of Figure 2.7.*

CHAPTER 3

Image-based geolocalization in natural environments

Image-based techniques and, in particular, image-to-terrain ones are suitable for both large scale visual geolocalization and camera orientation estimation when dealing with images taken in natural mountain environments. Such problem can be tackled by computing the alignment between the skylines extracted from the terrain, which is computed from the Digital Elevation Model (DEM) data, and mountain images. Heuristic methods based on edge detection work well on images taken in good conditions, but present difficulties with bad weather or occluded scenarios. In these cases, a cloud, a high voltage cable, a person, or a roof can impact negatively on the heuristic edge filter, e.g. a cloud edge can be treated as skyline and the mountain slope below would be erroneously regarded as noise.

This chapter presents the implementation process of a robust pixel-wise mountain skyline detection component, from the collection and preparation of the data set for training the Fully Convolutional Network [140] to the evaluation of the trained models; the model is able to detect skyline occlusions, thus improving the subsequent alignment w.r.t. the virtual terrain and is suitable for deployment on both server and mobile devices.

This chapter includes material from the following publication, co-authored by the candidate: [65].

3.1 Requirements

The tackled problem of robust skyline detection for the registration of images taken in natural mountain environments w.r.t. the terrain virtual panorama is a non-trivial task, which is subject to the following essential requirements:

- **Accurate skyline detection.** The skyline detection must be precise and

identify the portion of the topmost boundary edges of the mountain slopes that intercept the sky. Additionally, it must also cope with changing and complex visual conditions, such as variations in season, weather, illumination and vegetation.

- **Occlusion identification.** The designed solution must work in uncontrolled conditions, often characterized by the presence of irrelevant objects. Occlusions interrupting the natural horizon must be identified and treated appropriately, to prevent false alignments with irrelevant objects (e.g. mistaking a rooftop, a tree, or a person's head as part of the mountain skyline).
- **Server and mobile support.** The complete pipeline is meant to be executable on both server and mobile environments, and must thus overcome underlying technical constraints, given that mobile application development imposes numerous restrictions on the supported architectures, frameworks and libraries.

3.2 Data set collection

Publicly available data sets, such as [25] [44] [97] [193] are suitable candidates for training CNN models for sky/terrain segmentation. However, as discussed in Section 2.1, they were not built to deal with skyline interruptions, due to e.g. clouds, trees, buildings, people, etc., which can compromise the correct alignment w.r.t. the terrain model and thus the geolocalization of the image. To the best of our knowledge, no data set expressly designed for mountain skyline detection in presence of occlusions exists, and consequently we conducted an internal crowdsourcing campaign to create it.

The procedure to build the data set consisted in (1) a semi-automatic selection of images from web media content [32]; (2) a first crowdsourcing task to filter inadequate images (e.g. non mountain images identified incorrectly in step 1); (3) a second crowdsourcing task to annotate the skyline in the images and (4) final inspection and cleansing of annotated images.

The semi-automated selection of images, as explained in detail in [32], was performed by searching geo-tagged and user-generated photos in Flickr in a region of 300 x 160 km in the European Alps and by crawling images from touristic webcams in the same area. Both sources present common information: location, date and time of capture, but they differ in the nature of the pictures; while Flickr photos tend to be nicer, with a clear skyline, good weather conditions and from diverse points of view, webcam images tend to be more complex. Each webcam provides a temporal series of pictures, from which we sampled images with different weather conditions at different times of the day.

The gathered images were used as input for the web-based crowdsourcing platform visualized in Figure 3.1. Such tool supported the execution of two tasks:

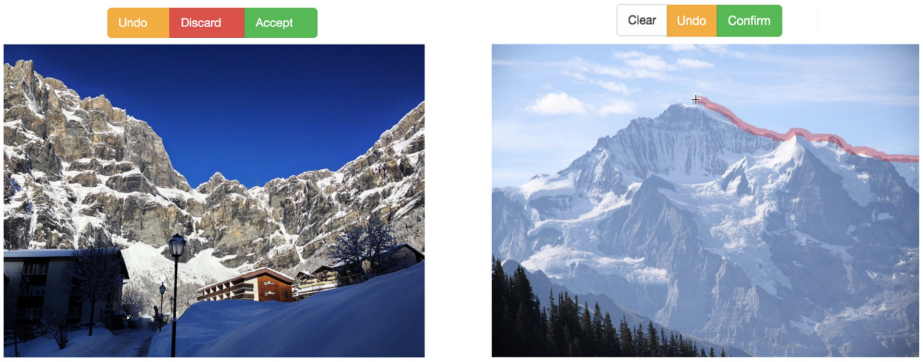
- **Mountain image selection.** The input for this task is an image and the output is a Boolean value specifying whether the image contains a mountain profile or not. An introduction to the problem and the output required were provided to the participants and criteria to discard images were explained in

	#	%
Occluded images	4,327	48.86%
Non occluded images	4,529	51.14%
Occluded columns	485,920	8.80%
Non occluded columns	5,038,444	91.20%

Table 3.1: *Dataset class distribution. Half of the images present non-continuous skyline. Less than 10% of columns are occlusions.*

a tutorial published in the home page of the task. Users were requested to discard images without mountains, with the skyline almost totally occluded, taken from a high point (e.g. from the airplane), and with applied filter effects.

- **Skyline image annotation.** The input of this task is an image and the output is a line (or a set of segments, if the skyline is interrupted by obstacles) that identifies the profile of the mountain. Given the image, the user can draw the skyline line, clear the annotation to erase the current annotation and restart drawing; and confirm the annotation.



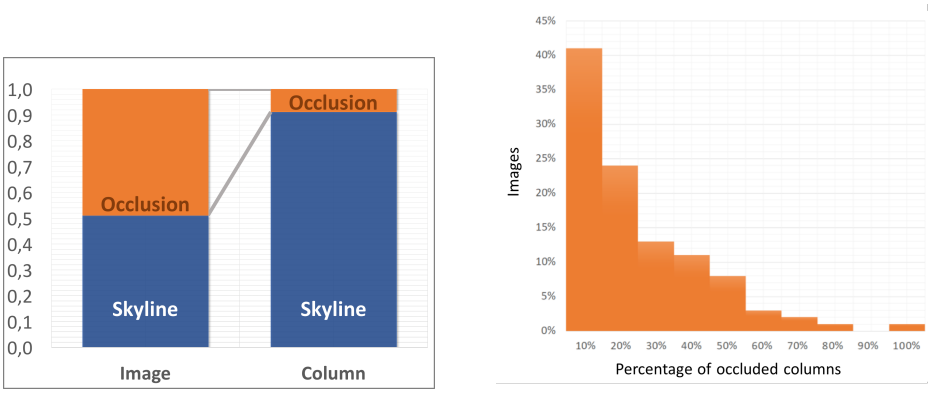
(a) *Mountain image annotator* **(b)** *Skyline image annotator*

Figure 3.1: *Crowdsourcing mountain skyline platform annotator.*

An internal crowdsourcing campaign with 17 workers was conducted using the described tool for the creation of the dataset. In the *Mountain image selection* task 9,001 images were accepted and 2,863 were discarded. In the *Skyline image annotation* task all 9,001 selected images were annotated.

After annotation, a data cleaning step was applied to remove wrong annotations and images with less than 240 pixels of height. After the manual cleaning, 8,856 annotated images were retained in the dataset.

Table 3.1 shows the presence of occlusions in the data set: nearly half (48.86%) of the images contain occluded skylines and occlusions are mainly small: less than 10% of the data set columns are part of an occlusion. Figure 3.2 shows the results of the analysis of the percentage of occlusion columns in the images; moderate occlusion/skyline ratios prevail: $\sim$ 42% of the images contain up to 10% of occlusion columns, and $\sim$ 23% up to 20%.



(a) *Occlusion distribution* **(b)** *Occlusion histogram*

Figure 3.2: *Dataset Class Distribution. Occlusions form patterns: the distribution over relative orientations of occluder-skyline is highly peaked around one mode.*

For each experiment, we performed a *holdout* segmentation; the dataset D was split into the training set D_{train}, the validation set D_{val}, and the test set D_{test}. We then trained and evaluated our models with an `80-20` setting, where 80% of the images were used for training and validation (64% and 16% respectively) and the remaining 20% for testing.

3.3 Pipeline for skyline detection

The goal of this part of the work is to implement a component for extracting the skyline from input images, portraying mountain landscapes in such a way that any irrelevant object occluding the skyline is identified and its profile discarded from the result. This task is accomplished by the two-step pipeline shown in Figure 3.3.

The pipeline comprises two modules: a pixel-wise skyline detector extracts the contour of the mountain; a column-based occlusion detector refines the classification by identifying the segments of the contour lines that are due to occluding objects. Figure 3.9 shows an example of the impact of occlusion removal.

3.3.1 Pixel-wise skyline detection

The pixel-wise skyline detector adopts the CNN architecture presented in Figure 3.4, which is an adaptation of the well known LeNet model [130]. The main differences are that: 1) odd sized 29x29 RGB input images are used instead of 28x28 gray-scaled ones, and 2) the output consists of only two classes, which represent whether the center pixel of the input image is part of the skyline (1) or not (0). As in LeNet, we consider the probability of a pixel to belong to each class. Moreover, the Fully Connected layers are replaced by Convolutional layers, to use the model as a Fully Convolutional Network (FCN) [140], which can take in input images of any size (bigger or equal to the model input size) and output a probability map with a value for each pixel. Multiple scale resolution objects are automatically tackled by the fact that the model is trained with images taken

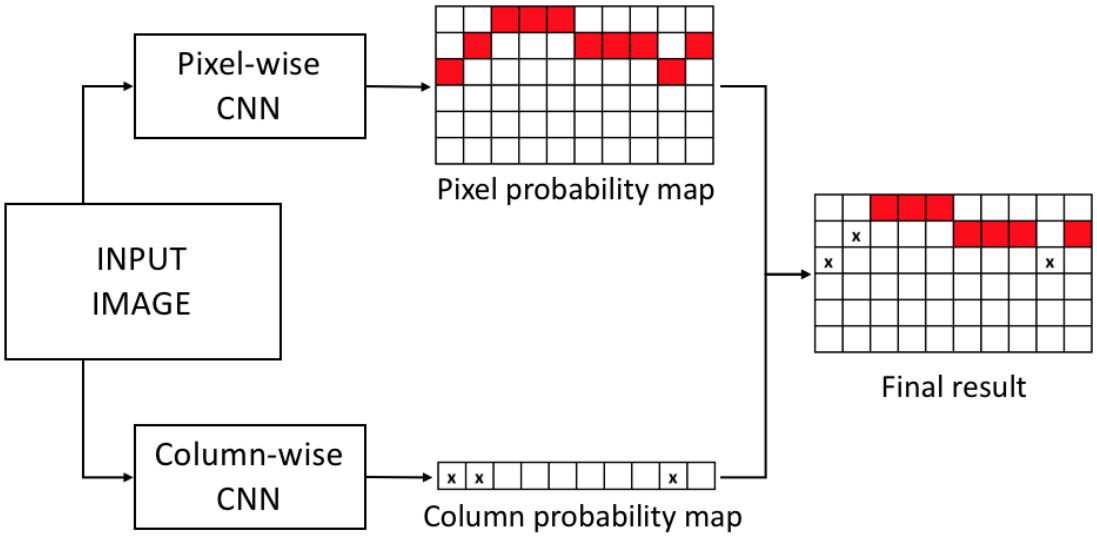


Figure 3.3: *Architecture of the skyline detection pipeline combining the pixel-wise and column-wise CNNs.*

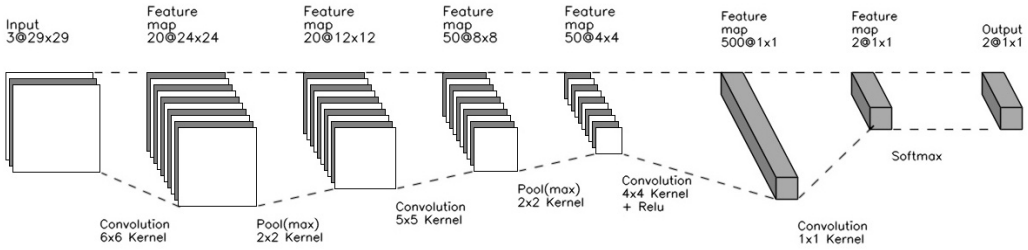


Figure 3.4: *Pixel-wise skyline detection CNN architecture*

from very variable distances.

The preparation of the data set involved the normalization of the mountain images and of the corresponding annotations, obtained by scaling down both to a predefined width (321 pixels) and then dilating the thickness of the annotation line(s) to a predefined size (9 pixels), to cope with small imperfections in the manual drawing of the skyline. The choice of the image width size is further discussed in Section 7.3.

Next, the heuristic extraction of positive and negative patches from the annotated mountain photos is performed. To obtain patches with the most informative content, we applied a very soft Canny filter to each image so as to compute the edge map, and selected as candidate patches only the ones with an edge pixel at their center. Patches were then labeled as positive or negative based on their central pixel: if it matched an annotated pixel, the patch was considered positive; otherwise, it was considered negative. Patches were partitioned based on the image subset they belonged to (training, validation and test), as specified in Section 3.2. That is to say, patches extracted from images in the training set were considered for training, and the same applied for validation and testing.

Since non-skyline pixels are much more numerous than skyline pixels, we generated an imbalanced data set by extracting 100 positive and 200 negative patches

per image. Two extraction strategies were evaluated:

- **Random sampling.** The set of all possible positive and negative patches was created, randomly sorted, and truncated to a maximum predefined number (100 positive and 200 negative samples).
- **Grid sampling.** To make the distribution of patches more uniform, each image was divided into a 3x4 grid and patches were extracted by iterating over the cells, picking only candidate patches with a minimum distance (3 pixels) from an already selected one. Then, 100 positive and 200 negative elements were selected from the candidate patches generated in the above-mentioned way.

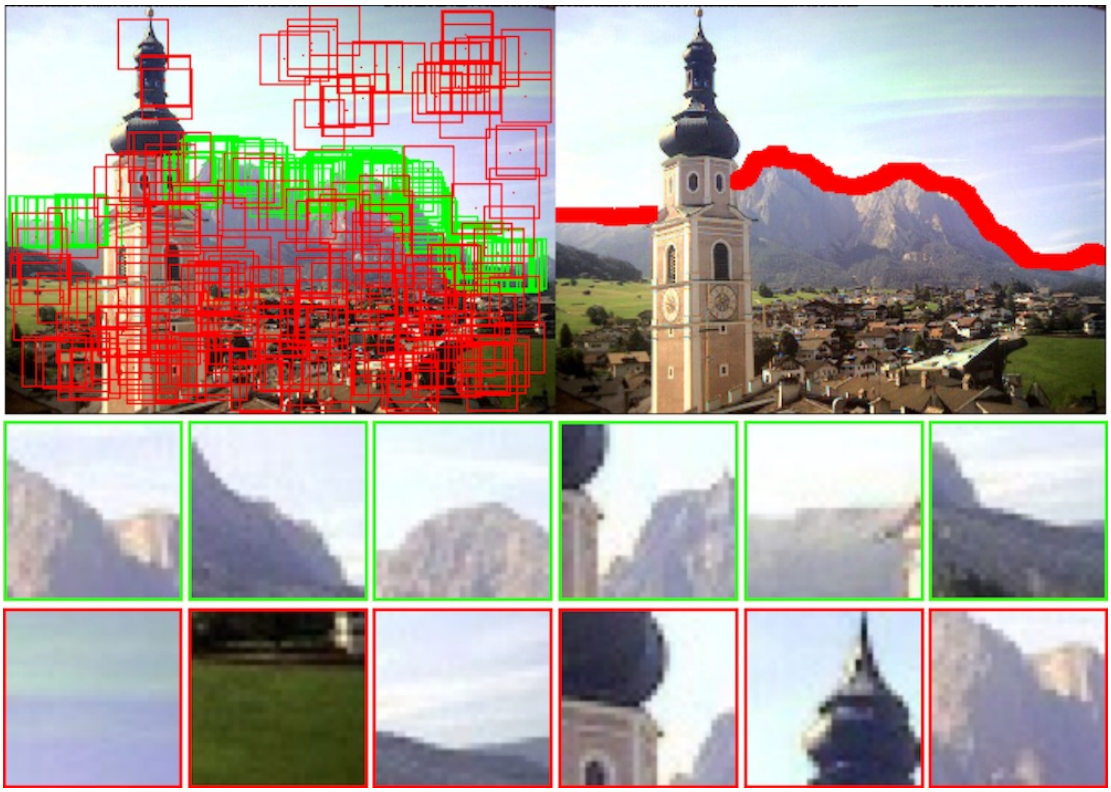


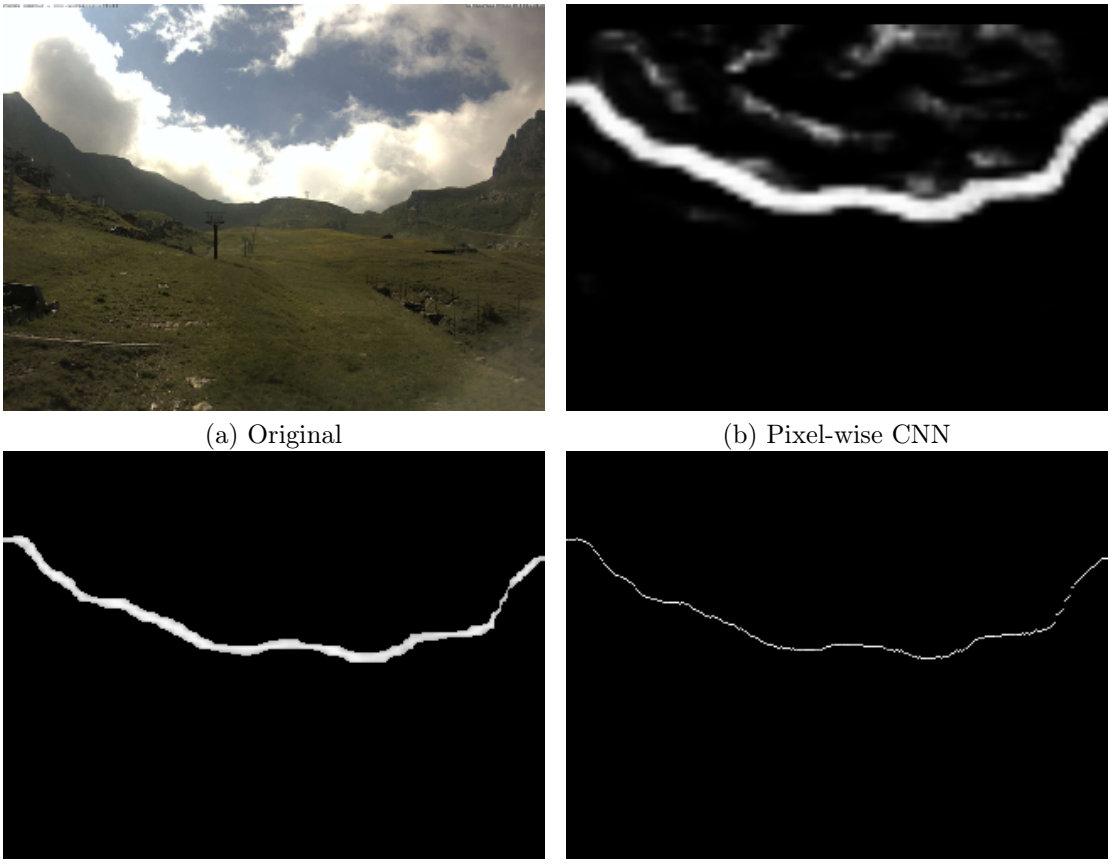
Figure 3.5: *Random sampling of patches for pixel-wise skyline detection. Green patches correspond to positive class and red patches to negative class.*

Figure 3.5 visualizes an example of the random sampling of patches applied to an image: green boxes correspond to positive patches and red boxes to negative ones.

Different positive to negative ratios were tested and 1:2 was chosen as the proportion providing the best outcome, and thus was the one employed in the evaluation described in Section 7.3.

The model was trained using the Caffe framework [112] and the total number of learned parameters of the resulting model was 428,732. At execution time, the FCN was fed with an entire image and returned a spatial map, in which each pixel was assigned the probability of belonging to the positive class. Such probability values were rescaled from the $[0 \dots 1]$ to the $[0 \dots 255]$ range.

The output of the CNN was post-processed to enhance the result. Figure 3.6 shows all the steps for computing the output: the original image (a) was processed by the CNN, obtaining the raw output (b). Then, a small erosion and a threshold were applied, whereby pixels with scores lower than a predefined value were removed (c). Such predefined threshold value was calculated to maximize the accuracy of the CNN and is further discussed in Section 7.3. Finally, a morphological skeletonization process was applied and at most 1 pixel per column was left (d). Diverse post-processing heuristics have been assessed and the described approach was the one that achieved the best results.



(a) Original (b) Pixel-wise CNN

(c) Erosion + Threshold (d) Skeletonization

Figure 3.6: *Pixel-wise skyline detection pipeline. Original image (a). Raw probability map output computed by the pixel-wise CNN (b). Probability map after erosion and thresholding (c). Result after morphological skeletonization and selection of one pixel per column (d).*

3.3.2 Column-wise detection

The pixel-wise skyline detection approach performs well and can cope, to a limited extent, with the presence of some skyline occlusions, e.g. due to clouds. However, the construction of the training set does not identify occluding objects explicitly and thus both the random and grid sampling procedures extract only a small number of negative patches that include common occluding objects. In addition, 29x29 squared patches may not always contain enough context to accurately pre-

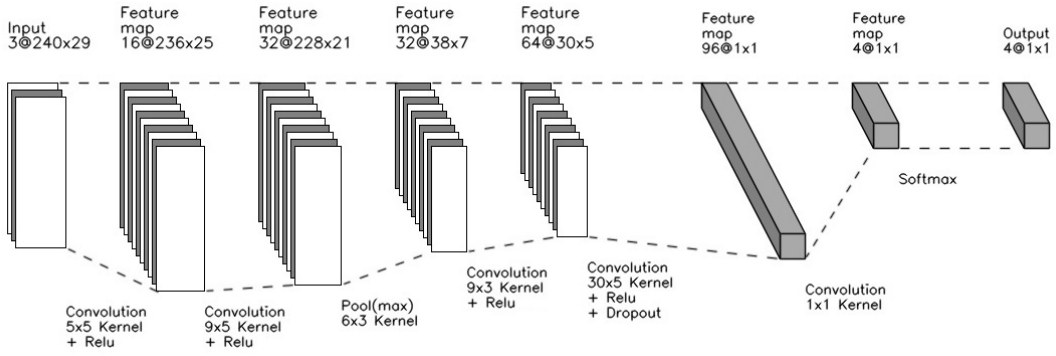


Figure 3.7: *Column-wise skyline detection CNN architecture*

dict obstacles. To address the presence of occlusions more specifically, we trained an additional CNN, using column patches instead of squared patches. The idea is that columns with a height equal to that of the input image provide better vertical context, which could help detect objects standing in front of the skyline, such as tree tops, people's heads, and bell towers. As shown in Figure 3.3, the column-wise CNN is executed after the pixel-wise CNN to filter out false positive skyline pixels corresponding to the profile of non-mountain objects.

The use of a column-based CNN is an alternative to sky/terrain semantic segmentation approaches, such as [3] [12] [13] [169], which could also indirectly support the identification of occlusions. The advantage of the proposed architecture is that the training of the column-based CNN component exploits the same annotations employed to build the training set of the pixel-wise skyline detector, eliminating the need of yet another (costly) image annotation campaign.

Figure 3.7 shows the architecture of the column-wise occlusion detector, which consists of a FCN that takes in input an image, resized to the predefined height of 240 pixels, and returns in output a vector of the probability value of each column to belong to the positive class (i.e., skyline). Note that the model only predicts whether the columns contains the skyline or not, but cannot determine the row position at which the skyline pixel appears in the column.

The data set preparation consisted in the extraction of 29x240 column-patches from the original size mountain images; 87.40% of the data set images have a height of at least 360 pixels, thus affording enough space to crop column patches in diverse regions.

The column patches were labeled automatically by reusing the same annotations employed to extract square patches for the pixel level skyline detector. To compensate the lack of explicit semantic labels for occluding object pixels, the extracted column patches were heuristically assigned to multiple classes, to make the CNN learn how to discriminate among negative samples of different nature, among which are the potential occlusions.

Given an image I and its ground truth annotation GT, we define:

- Skyline column: a column $I[\_, j]$ for which there exists an i such that the pixel $GT[i, j]$ is annotated as skyline.
- Non-skyline column: a column $I[\_, j]$ for which the pixel $GT[i, j]$ is annotated

as non-skyline, for all values of i.

Then, a column patch is defined as a rectangular 29x240 region of the image, labeled as follows:

- **Positive.** At least one pixel of the center column of the patch matches a pixel of the skyline annotation.
- **Negative-sky.** All the pixels of the center column of the patch lie above the skyline annotation.
- **Negative-terrain.** All the pixels of the center column of the patch lie below the skyline annotation.
- **Negative-occlusion.** All the pixels of the center column of the patch lie within a candidate occlusion region of the image; such region is a rectangle of parametric height defined as follows: : 1) it overlaps at least one non skyline column of the image; 2) it is delimited on the left either by the image leftmost column or by a skyline column; 3) it is delimited on the right either by the image rightmost column or by a skyline column; 4) all the image columns overlapped by the candidate occlusion region are non skyline columns.

Figure 3.8 visualizes a subset of the column patches of different classes extracted from a sample image.

Diverse data set configurations and ratios between the patch classes have been assessed. In the end, the best results, discussed in Section 7.3, were obtained by extracting 128 positive column-patches and 384 negative column-patches ($\frac{1}{3}$ for each negative subclass). The model was trained with the Caffe framework and includes 1,001,732 learned weight parameters.

To use the output of the column-wise CNN for eliminating false positives produced by the pixel-wise CNN, we summarize it into only two classes: positive and negative, summing up the probabilities of belonging to the negative-sky, negative-terrain and negative-occlusion classes.

Figure 3.9 shows an example of the complete CNN pipeline: the original image (a) is first processed with the pixel-wise CNN (b) and subsequently processed with the column-wise CNN (c). The second CNN discards the false positives created by the first CNN, which misclassified tree and roof tops.

3.4 Evaluation

The maximum accuracy achieved by the pixel-wise CNN model over the test data set *at patch level* is 95.81%, obtained with a threshold value for positive probability of 124; the maximum accuracy at *at column-patch level* for the column-wise CNN is 89.06%, with a threshold value of 143. However, accuracy measured at the patch level does not represent the quality of the output for the extracted skyline well, be it continuous or interrupted. Therefore, we defined quality metric functions that compare column by column, the skyline extracted by the detectors w.r.t. the one manually annotated in the ground truth.

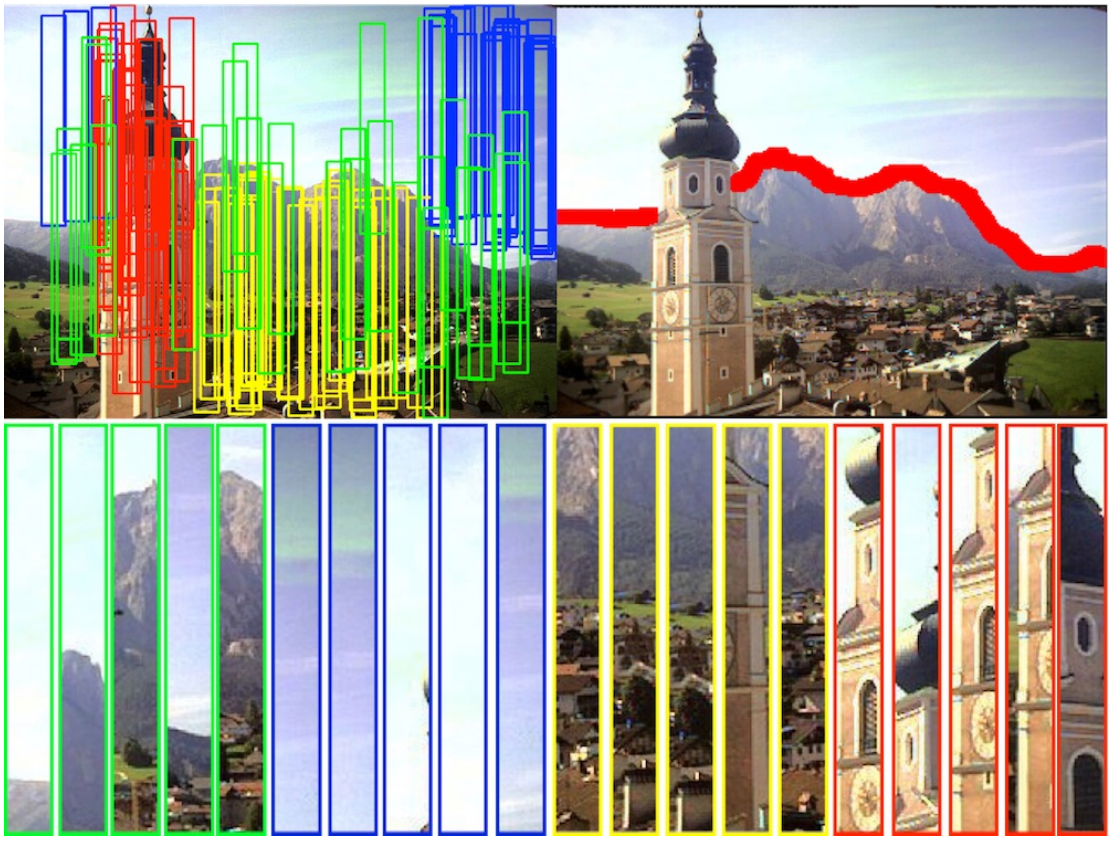


Figure 3.8: *Example of column-patch extraction. Green column-patches correspond to positive class, blue to negative-sky, yellow to negative-terrain and red to negative-occlusion.*

3.4.1 Experimental setup

The evaluation of the skyline detectors was performed by running a series of evaluation rounds, in which the proposed metrics were computed for the pixel-wise CNN and the combined CNN over three subsets:

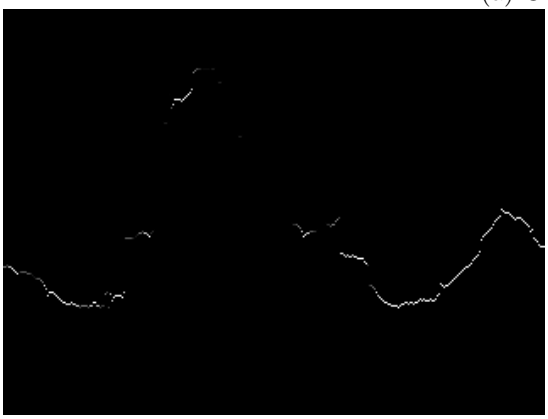
- **Complete.** It contains the whole set of 1,771 images with both continuous and interrupted skylines, which account for 100% of the test images.
- **Any Occluded.** It contains 866 images with at least one interrupted skyline column, which comprises 49.9% of the test images.
- **Occluded($\leq$30%).** It contains 674 images with at least 30% of their columns occluded and comprises 38.06% of the test images and 77.83% of the *Any Occluded* set).

Moreover, each of the images in the test set were resized to 240 pixels of height and their corresponding widths, so as to match the exact column-wise CNN expected input size.

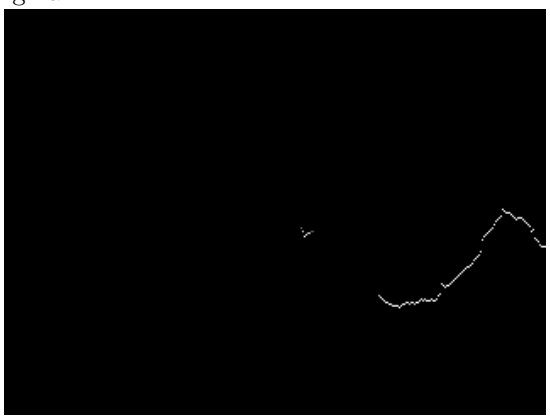
3.4.2 Metrics

Differently from the approaches in [2], [3] and [169], which do not consider occlusions and postulate that skylines are always complete and span the entire image, the proposed metrics do not measure the distance between the annotation and

(a) Original



(b) Pixel-wise CNN



(c) Column-wise CNN

Figure 3.9: *Pipeline combining pixel-wise and column-wise CNN: original image (a); output computed by the pixel-wise CNN (b); output computed by the combined pipeline (c).*

the estimated skyline, but instead assess whether *each skyline pixel* is correctly classified and then average such basic measure over the relevant image columns; the following metric functions are defined:

- **Average Skyline Accuracy (ASA)** measures the percentage of image skyline columns for which at least one of the estimated positive pixels matches one of the annotation pixels.
- **Average No Skyline Accuracy (ANSA)** measures the percentage of non skyline columns for which the CNN output does not contain positive pixels; this metric evaluates false positives in images with an interrupted skyline.
- **Average Accuracy (AA)** measures the percentage of columns in which the annotation and the estimated skyline agree, considering agreement when none contain pixels or at least one of the estimated positive pixels matches one of the annotation pixels.

Let $CNN(i, j)$ be a function that returns 1 if the image pixel at coordinates (i,j) belongs to the skyline extracted by the CNN (0 otherwise) and let $GT(i, j)$ be

a function that returns 1 if the pixel (i,j) belongs to the ground truth skyline (0 otherwise). Let *cols* denote the number of columns the the image(s) under test.

$$ASA = \sum_{j=1}^{cols} I_{GT \wedge CNN}(j) / \sum_{j=1}^{cols} I_{GT}(j) \tag{3.1}$$

$$ANSA = \sum_{j=1}^{cols} I_{\overline{GT} \wedge \overline{CNN}}(i,j) / (cols - \sum_{j=1}^{cols} I_{GT}(j)) \tag{3.2}$$

$$AA = \frac{1}{cols} \sum_{j=1}^{cols} I_{agree}(j) \tag{3.3}$$

where:
$I_{GT}(j) := 1 \; if \; \exists i |\; GT(i,j) = 1; \; 0 \; otherwise$
$I_{GT \wedge CNN}(j) := 1 \; if \; \exists i |\; GT(i,j) = 1 \wedge \; CNN(i,j) = 1; \; 0 \; otherwise$
$I_{\overline{GT} \wedge \overline{CNN}}(j) := 1 \; if \; \forall i |\; GT(i,j) = 0 \wedge \; CNN(i,j) = 0; \; 0 \; otherwise$
$I_{agree}(j) \; := 1 \; if \; I_{GT \wedge CNN}(j) = 1 \vee I_{\overline{GT} \wedge \overline{CNN}}(j) = 1; \; 0 \; otherwise$

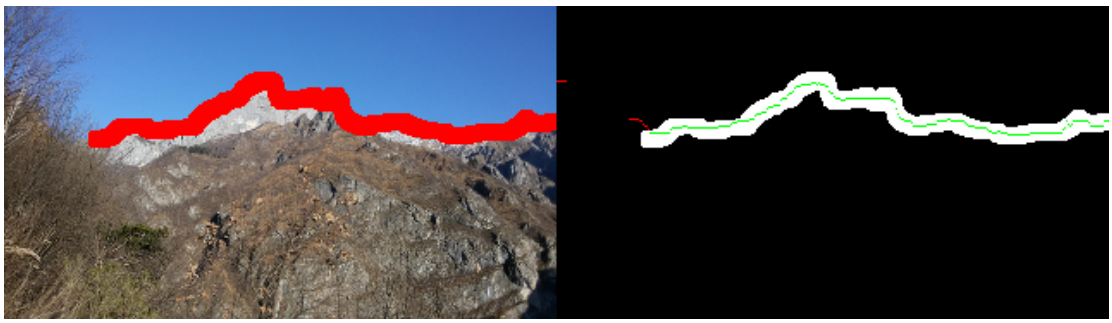


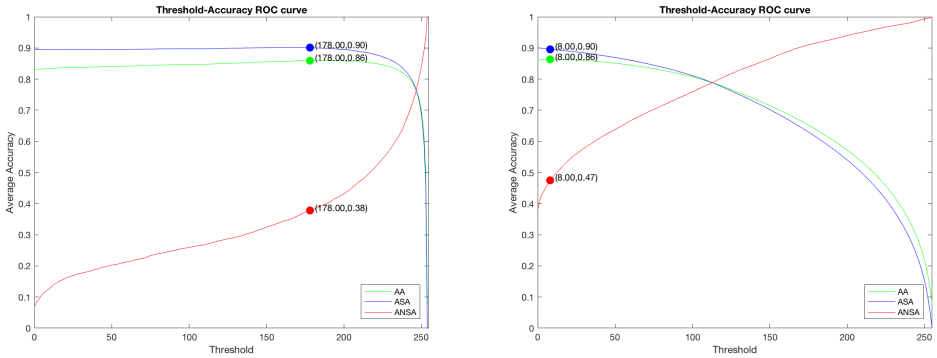
Figure 3.10: *Evaluation of an image with interrupted skyline and values of the metric functions. Average Skyline Accuracy: 98%. Average No Skyline Accuracy: 73%. Average Accuracy: 94%.*

Figure 3.10 illustrates an example of the metrics: a mountain image with the ground truth annotation is shown on the left, for which the quality metrics are computed on the output produced by the pixel-wise CNN. On the right, pixels in white represent the ground truth annotation, pixels in green represent correctly predicted skyline pixels, while pixels in red represent incorrect ones. The metric values for the image are: 98% of ASA, 73% of ANSA and 94% of AA.

3.4.3 Experimental results

Before evaluating the test data set, we ran an assessment over the validation data set, to estimate the threshold value that maximizes the AA metrics. AA was selected as the most relevant quality metric to maximize, because it represents the agreement between the ground truth and the extracted skyline over both occluded and non-occluded segments.

Figure 3.11 presents the evaluation curves of the pixel-wise CNN (a) and of the combined CNN (b) over the complete validation data set. For the combined CNN, Figure 3.11 (b) shows the accuracy curve used to define the threshold parameter of the column-wise component, when the best threshold value for the pixel-wise component is used.



(a) *Pixel-wise CNN threshold parameter analysis.*

(b) *Column-wise CNN threshold parameter analysis, for the combined CNN with pixel-wise threshold set to 178.*

Figure 3.11: *AA curves computed on the complete validation data set to identify the best threshold values for the pixel-wise CNN (a) and for the column-wise component of the combined CNN (b).*

The pixel-wise CNN achieved an AA of 86%, ASA of 90% and ANSA of 38% for a threshold value equal to 178. The combined CNN, with the pixel-wise threshold set to 178, achieved an AA of 86%, ASA of 90% and ANSA of %47, for a threshold value of the column-wise component equal to 8. The low threshold value of the column-wise component means that columns are discarded only when they get extremely low probability of being positive.

The same analysis was performed for the *Any Occluded* validation set. The found optimal threshold values are then applied in the assessment of the *Any Occluded* and *Occluded(≤30%)* test set.

Table 3.2 reports the results of evaluation on the three test sets, with the threshold values determined from the analysis on the validation sets.

	Pixel-wise CNN			Combined CNN		
Set	ASA	ANSA	AA	ASA	ANSA	AA
Complete	89.82%	34.56%	85.42%	89.34%	43.78%	85.72%
Any Occluded	86.21%	43.27%	79.26%	84.51%	55.30%	79.78%
Occluded(≤30%)	86.86%	46.96%	83.15%	86.26%	55.08%	83.33%

Table 3.2: *Skyline image-level evaluation metrics for diverse test data subsets computed with the pixel-wise CNN and combined CNN.*

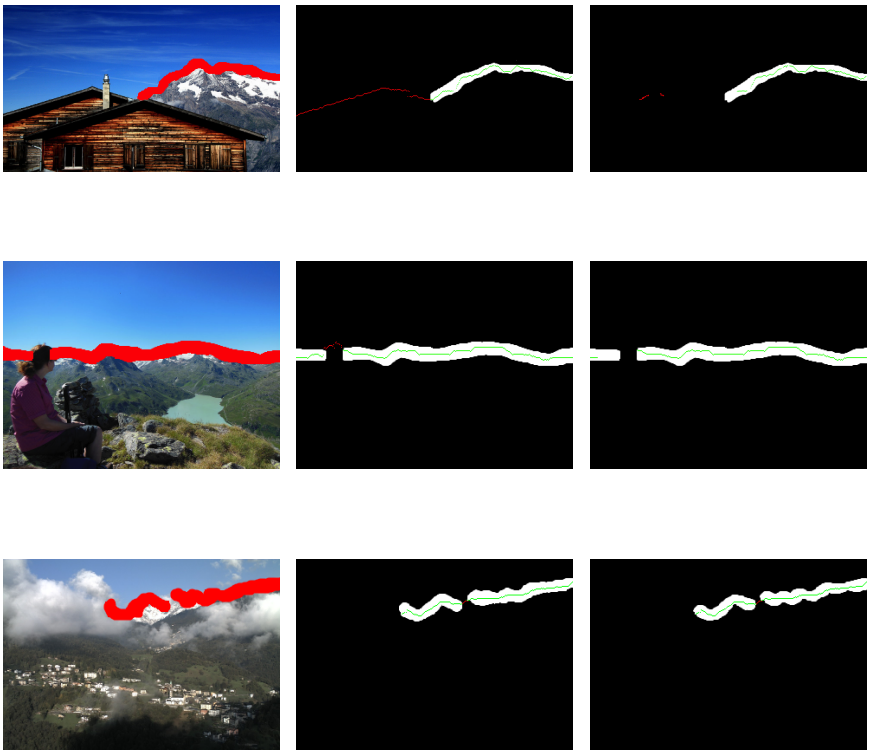
The pixel-wise CNN achieved reasonable results for ASA of more than 86% in all subsets and consequently behaved well for AA, due to the fact that columns with skyline are highly predominant and thus ASA highly influences the overall performance. On the other hand, the pixel-wise CNN exhibited lower ANSA values, which could be expected because occlusion patches are underrepresented in the training set. The combined CNN improves the ANSA values by approximately 10%, which is the contribution of the more significant context provided by columns, instead of patches, in the training. As a consequence, also the AA

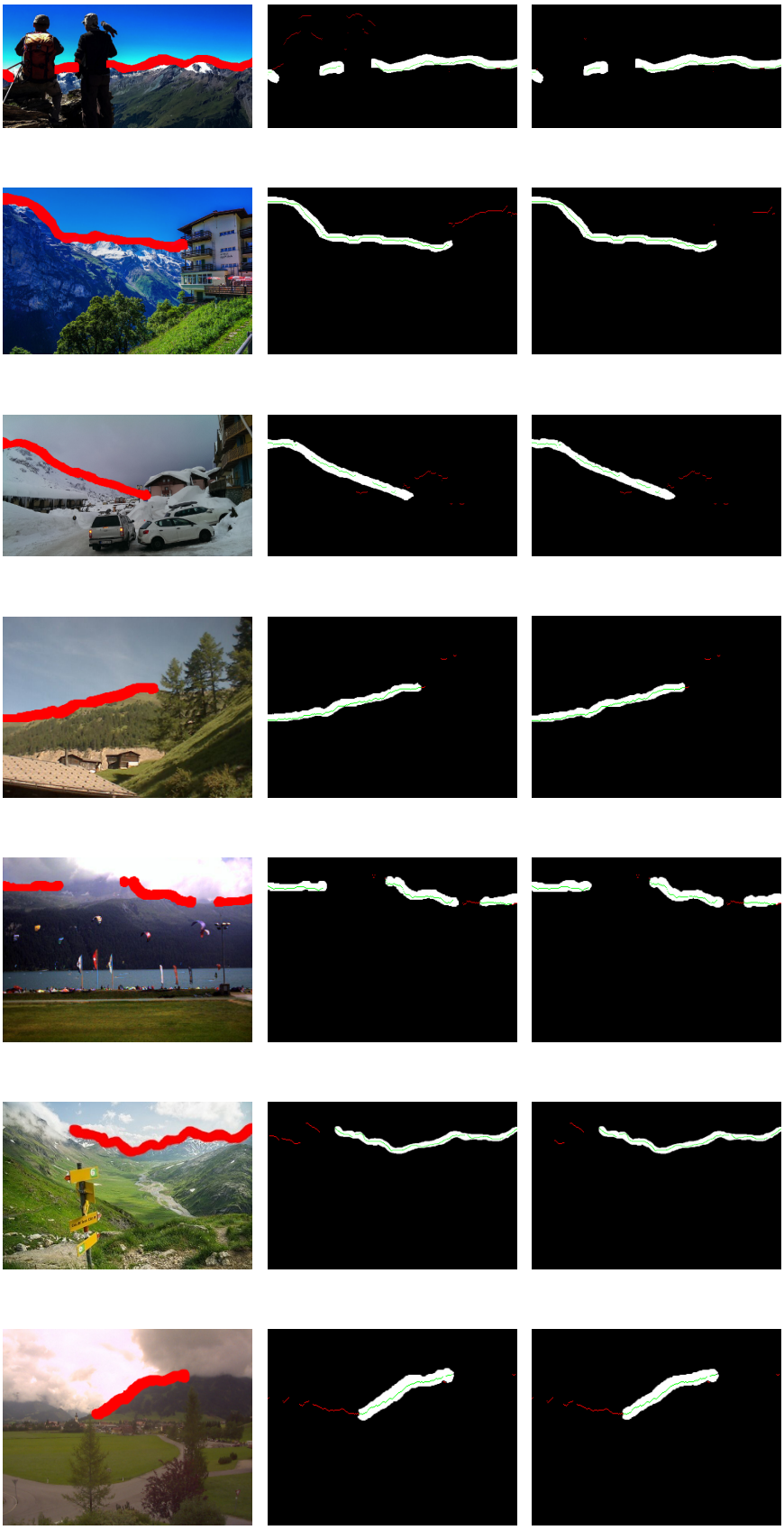
slightly increased in all the test sets, which means that the combined model handles occlusions adequately, without impacting negatively on the detection of the visible portions of the skyline.

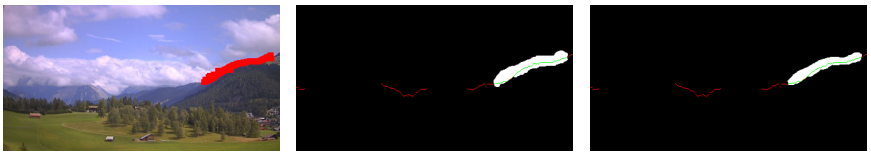
Results are better for the *Occluded(≤30%)* test set than for the *Any Occluded* one, which is explained by the fact that no skyline columns are underrepresented and more difficult to detect.

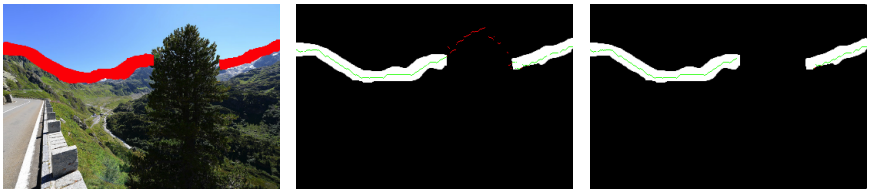
Finally, we observe that a ≈10% improvement of the ANSA metrics impacts the ability of discarding false positive segments of the mountain profile. This may have a high impact on the user-perceived quality of the augmented reality application that uses the extracted skyline to identify the peaks *that are in view*.

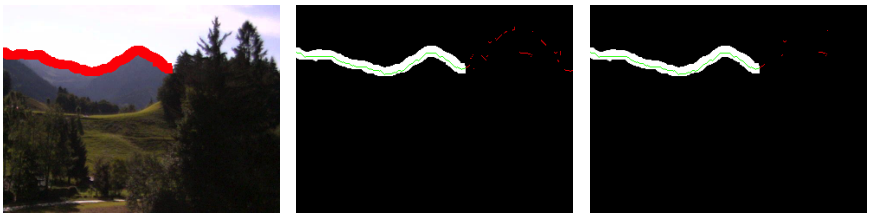
Several examples of the improvement of the combined CNN over the pixel-wise CNN in images with a variety of different occlusions are presented next. The left column shows the ground-truth with the skyline annotated in red, the middle column shows the results obtained using the pixel-wise CNN and, finally, the third column shows the results obtained using the combined CNN. All skyline pixel matches are shown in green, the annotation is shown in white and the occlusion wrong classifications are shown in red.











CHAPTER 4

Outdoor Mobile Augmented Reality Framework

Outdoor Mobile Augmented Reality applications project information of interest onto views of the world in real-time. Their core challenge comprises the recognition of the meaningful objects present in the current view and the retrieval and overlay of pertinent information onto such objects. Capturing the user's attention has become increasingly difficult due to the enormous number of mobile applications available on the online markets. Therefore, providing satisfying, engaging and novel types of user experiences becomes a critical factor in determining the success and diffusion of a mobile application. Traditional AR applications rely on the position and orientation sensors of the device to estimate the location of the user and her field of view, irrespective of the content actually in view. Examples include sky map applications that show the names of constellations, planets and stars based on the GPS position and compass signal. An obvious limitation of these approaches is that they may provide information that does not match what the user is seeing well, due to errors in the position and/or orientation estimation or to the presence of objects partially occluding the view. Moreover, the development of outdoor Mobile Augmented Reality applications poses several technical challenges related to the acquisition, selection, transmission and display of information.

In this chapter, we report on the development of a framework for Outdoor Mobile Augmented Reality applications that aims at addressing the above-mentioned challenges. We define the underlying requirements, architecture, components and workflows, as well as a characterization for the dimensions of heterogeneous data used to augment the camera view of applications of this nature. Finally, we discuss the application to the development of the outdoor use case for the overlay of mountain peak information onto views of mountain landscapes.

This chapter includes material from the following publications, co-authored by

the candidate: [60] [66].

4.1 Requirements

To be effective, outdoor Mobile AR applications pose several technical challenges that make their development a non trivial task: 1. *Mobile environment.* The target mobile devices comprise strict energy consumption constraints and lower memory and computational resources w.r.t. Web multi-tier architectures. 2. *Uncertain positioning.* The position and orientation sensor errors make the location estimation potentially noisy; thus the identification of the relevant objects from these signals alone cannot be assumed to be fully reliable. 3. *Uncontrolled viewing conditions.* The objects to be identified may have no fixed, a priori known appearance, because the viewing conditions can drastically change due to weather, illumination, occlusions, etc. 4. *Uncertain internet connection.* Network connectivity can be unreliable, especially for rural and mountain regions, where even today internet coverage is patchy. .

Based on such presented drivers, the target AR systems must cope with the following requirements:

- **Accurate visual recognition.** They must understand the current visual environment surrounding the user by exploiting frames captured by the camera and thus, potentially correct other noisy sensor readings.

- **Sensor fusion.** They must compute the position of the augmented information by estimating and fusing sensor readings coming from multiple heterogeneous sources such as GPS, compass, magnetometer, gyroscope and camera.

- **Fast response time.** They must overlay augmented information in real-time, and no significant overhead for image processing initialization is acceptable as users do not tolerate delays in the order of seconds at every start of the app.

- **Pertinent information retrieval.** They must find the appropriate information pertinent to the user's current location, activity, interest and view.

- **Bi-dimensional reduction.** Although the objects' position in the real-world is estimated in the 3D space, the on-screen rendition requires a projection onto the 2D surface of the camera view, based on a model of the camera.

- **Proper view augmentation.** They must overlay the retrieved information onto the device screen in a way that is stable and adequate to the user's experience. This may also comprise user interaction by filtering elements based on certain properties, scrolling through pages, visualizing more details of a specific element or taking pictures with the augmented information.

- **Offline support.** They should ideally guarantee functioning, in spite of uncertain Internet availability.

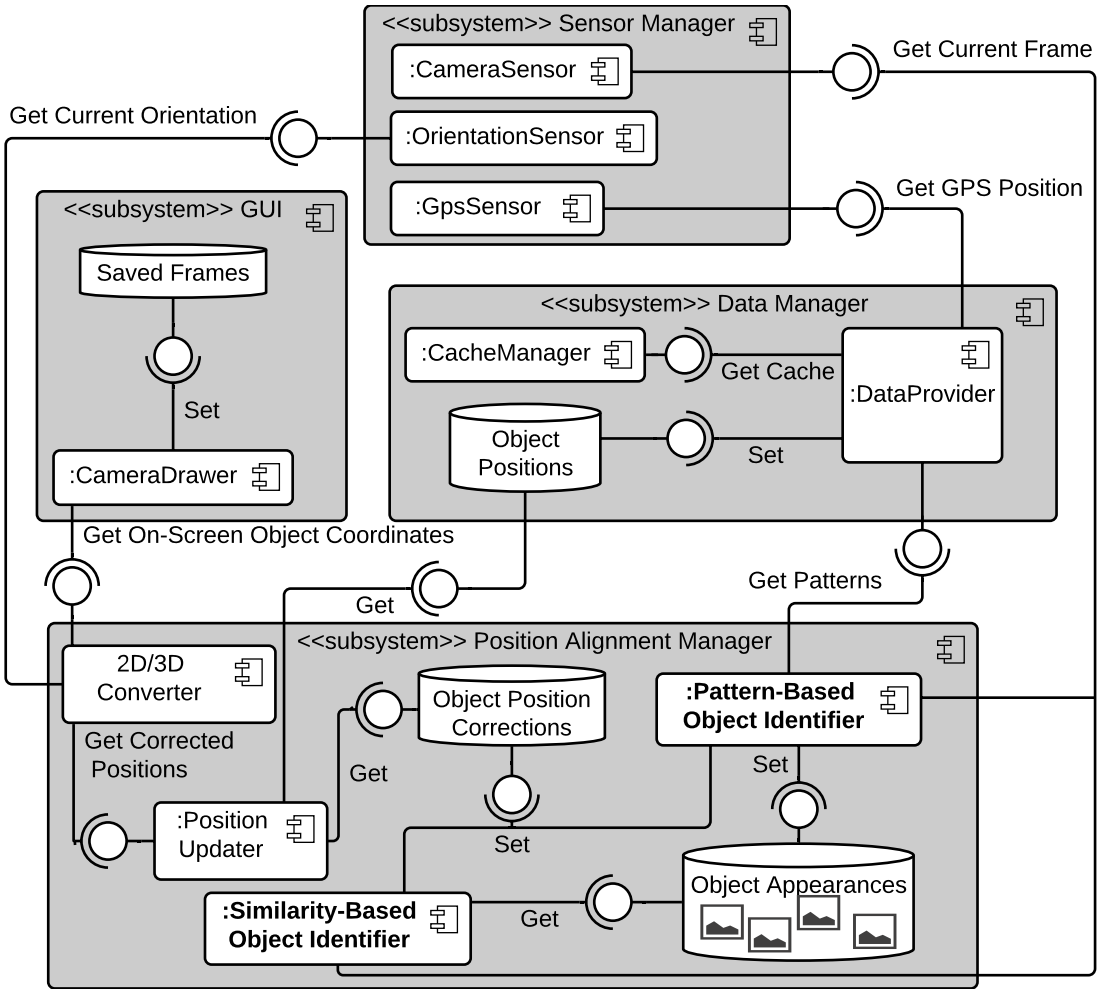


Figure 4.1: *The proposed architecture of a mobile outdoor AR application.*

4.2 The Development Framework

The framework proposed in this chapter aims at supporting the design of mobile AR applications for the enrichment of outdoor natural objects. Restricting the focus to devices that support a bi-dimensional view, a generic architecture must be realized. Such architecture must receive as first input a representation of the reality - in which the user is embedded - captured by the device sensors; such representation typically comprises a sequence of frames, captured by the camera at a fixed rate, and the position and orientation readings of the device, captured by the GPS and orientation sensors respectively. The second input to receive corresponds to the information regarding the possible objects present in a region of interest. Finally, the architecture outputs the on-screen positions of relevant objects and the association of relevant meta-data to such objects, computed at the same frequency of the input capture.

Figure 4.1 presents an UML component diagram that represents the reference architecture of a mobile outdoor AR application. The key idea is to enable the real-time augmentation process given a proper partition of functionality and a

mix of synchronous and asynchronous communications among the modules. The architecture consists of four sub-systems: the Sensor Manager, the Data Manager, the Position Alignment Manager and the Graphical User Interface (GUI).

4.2.1 Sensor Manager

The *Sensor Manager* coordinates data acquisition from the device sensors. It typically comprises one module per each signal processed by the application; the generic configuration comprises the *GPS Sensor Manager*, the *Orientation Sensor Manager* and the *Camera Sensor Manager*. The modules work asynchronously and provide input to the *Position Alignment Manager and Data Manager*, which subscribe to their interface and are notified when a new signal arrives from a sensor.

4.2.2 Data Manager

The *Data Manager* is responsible for providing the other sub-systems with the initial positions of the objects in view and the corresponding meta-data for enriching them. It receives as input the specification of an area of interest (typically inferred from the user's location), and interacts with an external repository containing a virtual representation of the world (e.g, a sky map or a DEM). It produces as output *Object Positions*, which specify the (initially approximate) 3D coordinates of the candidate objects to display. Within the *Data Manager*, a *Data Provider* component queries one or more external geo-referenced data sources, with the current user's location, and extracts the coordinates of the objects that are likely to lie within the view of the user. For example, in a sky observation app, it queries the sky map for the celestial coordinates, plus meta-data such as type, name, distance, etc., of the potentially visible objects. The *Cache Manager* implements data pre-fetching and synchronization policies, based on information about current cache content, network availability, and cost of data transfer. Since data about the objects can be large, the *Cache Manager* realizes a trade-off between on-demand transfer from external data sources and caching in the local storage of the device. Furthermore, it enables disconnected usage, as needed in the outdoor scenario, in which internet connection may not be always granted.

4.2.3 Position Alignment Manager

The *Data Manager* provides a fast computation of the initial *Object Positions*, to enable the immediate update of the GUI. But its output may be noisy, because the estimated user's position, the camera orientation and the virtual world representation may all contain errors. It is well-known that the GPS and orientation signal of mobile devices may be inaccurate; on the other hand, also the virtual world representation, e.g. a Digital Surface Model (DSM), may be affected by errors, e.g. due to low resolution. Therefore, the *Position Alignment Manager* comprises components for updating the positions of the objects, adapting them to the actual content of the camera view, and projecting them to the device's view. It takes in input the initial object positions provided by the *Data Manager* and produces in output the corrected on screen object coordinates. To support the

trade-off between accuracy and speed, the (demanding) computations required for improving accuracy are delegated to separate modules, which provide asynchronous corrections to the initial candidate positions, by applying content-based object detection techniques. These modules feed the *Object Position Corrections* store with the adjustments computed asynchronously, which the *Position Updater* and *3D/2D Converter* components exploit to correct the on screen coordinates used by the GUI. Examples of components for the content-based refinement of object positions are *Pattern-Based* and *Similarity-Based* Object Identifiers.
A *Pattern-Based Object Identifier* performs a frame-based match. It uses the virtual world representation as a pattern to search within the real-world image. It takes in input the virtual representation of the world (e.g. the synthetic rendition of a constellation or of a piece of mountain skyline) and computes a ranked list of approximate matches between the virtual image and the real one, w.r.t. some similarity function. This component can also exploit advanced sophisticated Computer Vision techniques, such as classification, segmentation, detection or computation of embeddings, in order to enhance final matching results. As a collateral output, the *Pattern-Based Object Identifier* can also extract from the real-world image the regions that correspond to the identified objects, according to the best match. Such artifacts, cached in the *Object Appearance Store* of Figure 4.1, denote the visual appearance of the objects of interest in the current view and can be used for accelerating the correction of objects' positions when the view changes.
A *Similarity-Based Object Identifier* performs object-based similarity search; it takes in input the object appearance artifacts and searches them in the frame, using computer vision techniques.
Finally, the *2D/3D Converter* projects 3D positions onto the bi-dimensional screen space. It takes in input the device position, orientation and Field Of View (FOV), applies a prospective projection, determines the on-screen coordinates of the candidate objects and discards those out-of-view, e.g, due to micro-movements of the device. For example, it projects the celestial coordinates of the relevant sky objects into on-screen coordinates.
The asynchronous communication between the components that compute position corrections and those that project positions and render the augmented reality view aim at enabling a best effort, near real-time adjustment of the view. The prospective projection is a constant-time procedure, so that the total response time of the *Position Updater* and of the *3D/2D Converter* is linear w.r.t. the number of candidate objects. Since this number is reasonably bound, the resulting time complexity is constant, which allows the mobile device to call the *Position Updater* and the *3D/2D Converter* synchronously at every frame arrival and redraw the view in near real-time based on the best available approximation of the object positions.

4.2.4 Graphical User Interface

The *Graphical User Interface* (GUI) receives the on-screen bi-dimensional object coordinates from the *2D/3D Converter* in a constant-basis and draws on top of the camera view the indicated objects, thus displaying their corresponding meta-data

in the computed coordinates.

4.3 Dimensions of heterogeneous augmentation data

The engagement potential of an AR application depends on its utility for the user, which in turn depends on the quality, timeliness, relevance and usability of the information displayed on the screen. The management of the augmentation data collected by the geo-object acquisition step can be characterized by the following dimensions.

Object semantics.

This dimension characterizes the purpose of the core objects published for the user. In a touristic application, the core objects would focus on touristic monuments or landmarks , which serve an identification and an orientation purpose; secondary geo-referenced objects can also be relevant (e.g. scenic views, shelters, cultural heritage spots, touristic info points, and local events) and serve the purpose of improving the utility of the application. In the design of an AR application with multiple objects of interest of different kinds, it is important to define a priority ranking among objects; provided that the screen space is limited and the interaction of the user during an outdoor activity must be minimized, it is imperative to show the most relevant objects first.

Object provenance.

Objects may come from a single source or from multiple ones (e.g. a landmark's information may be spread in more than one GIS). Multiple sources pose classical heterogeneous data management challenges, including the reconciliation of object properties with different values in alternative data sources, such as the use of alternate names or differences in the geographical coordinates or boundaries for the same object. Given the tight response time constraints of mobile AR apps, reconciliation should be performed offline in the geo-object acquisition step. Furthermore, the usage policies of the data sources collected must be verified in order to ensure that their data are publicly available or exploitable under agreement.

Object storage and availability.

Most modern mobile apps are cloud-enabled and rely on data storage at the back-end and on Internet connectivity between the client and the server. However, an online connection cannot be taken for granted, because in outdoor conditions Internet connectivity may be absent for certain areas. Therefore, policies for objects storage and availability must be introduced to ensure both online and offline functioning. Such policies must address the download of the data required for disconnected usage, which in turn entails the segmentation of the information to cope with the cases in which the size of the entire data set exceeds the storage capacity of the phone or the complete download would result in unnecessary data transmission cost. Alternative solutions are possible: data can be segmented beforehand into fixed-size modules (e.g. countries, regions, cities and mountain ranges), or the user may be allowed to define custom areas dynamically. Caching

data locally at the client side poses classical cache management problems [34]: cache expiration, validation, and replacement rules must be enacted to ensure that the arrival of new data (e.g. more peaks available in a data source) are promptly reflected in the cached copy. Semantic cache transparency requires the user to be informed when the local copy of the data becomes outdated, so as to allow him to decide whether to tolerate misalignment (e.g. when he is in the middle of an outdoor activity) or execute a cache update (e.g. if the notification arrives when at home). As usual in cache management problems, the most appropriate policy depends on the size of the cached data, on the frequency and granularity of an update, and on the tolerability of misalignment. The object semantics clearly influence the latter aspect: landmarks information varies slowly and mostly in an incremental way, so that the cost of misalignment is low; whereas local event information may prevent caching at all. Besides the cache maintenance issues, local data storage also entails code mobility problems [165]: if the data are subjected to intensive processing before their rendition, a balance must be found between offline data usage and the performance overhead of processing data at the client side, which may be unfeasible or incompatible with the real-time usage requirements of a mobile AR application. E.g. caching DEM data at the client side may enable the application to work also in disconnected mode, but requires sufficient computation power on the phone for rendering the 3D point cloud in real-time. Deciding where to execute the code opportunistically may result in the best trade-off between conflicting requirements.

Object data compression.

When the data to be cached in the device are sizable, compression may reduce transmission time and storage occupation. E.g. DEM data consists of numerical data in massive quantity, which can be effectively compressed before transmission and caching [33]. The selection of the compression algorithm must evaluate the trade-off between the compression rate and the decompression overhead, which must be acceptable in a mobile application where low response times can be detrimental for the user experience.

Object media type.

The relevant objects may have different formats: label, text, icon, URL, or a mix thereof. In outdoor applications, text and images should be used sparingly and preferably not overlaid on the real-time camera view, because they may mask a too large portion of the real view. On the other hand, fragmenting the user interfaces in too many screens may also induce usability problems, because the application is used in conditions where switching among screens may be difficult (e.g. while walking or with strong light that makes screen reading and command execution difficult).

Object visualization.

The most appropriate visualization method depends on the semantics and media type of the object. Objects can be displayed as points (e.g. stars), 2D areas (e.g.

cardinal direction) and polylines (e.g. trails), possibly accompanied with icons and labels to convey some prominent attributes (e.g. an icon to suggest the object's semantics and a label to display the name and essential features). Visualization design must balance information content and visual clutter. When the relevant objects are too many to fit in the screen, policies must be designed to display them selectively based on their relevance; technical methods for doing this could be pagination and scroll mechanisms or filters based on object properties. Also in this case, interaction design should allow the user to obtain the required information with minimal gestures. When object selection mechanisms are provided, they must be accompanied by configuration option, which provide reasonable defaults that apply without any user's intervention. A pagination and scrolling mechanism must be accompanied by a default policy for deciding the objects that appear in the initial page: e.g. the most salient or closest elements. A filter should be accompanied by the default value of the filtering properties: e.g. the maximum distance to the objects to show.

4.4 Outdoor Mobile AR application for mountain exploration

In this section, we discuss how the general framework for Outdoor MAR described in the preceding section has been applied to the development of a MAR application for mountain exploration, with a particular focus on the studied use case.

4.4.1 Framework instantiation

The architecture of Figure 4.1 has been instantiated to the mountain exploration use case and required the adaptation of certain offline algorithms, defined in SnowWatch project [61] for their deployment on mobile devices. In the sequel, we describe the application-specific concepts and component refinements introduced for the mobile context.

The *Objects* to be identified are mountain peaks and the *Object Positions* are 3D global system coordinates situated in a unit sphere centered in the device location.

An application-specific *Cache Manager* has been implemented, responsible for pre-fetching and caching the DEM fragments corresponding to the geographical region the user is visiting. Pre-fetching is enabled when the Internet connection of the device is enabled and cache data are used by the *DataProvider* component to compute the *Object Positions* during outdoor usage. When the user moves out of the region for which data are in the cache, a cache miss triggers the download of a new fragment, which, in case of cache full, replaces the fragment relative to the region visited earliest.

The *Similarity-Based Object Identifier* component is implemented with a state-of-the-art cross-correlation patch recognition technique [27], which has been ported to the mobile execution environment.

The component where the most relevant adaptations have been introduced is the *Pattern-Based Object Identifier*, described next.

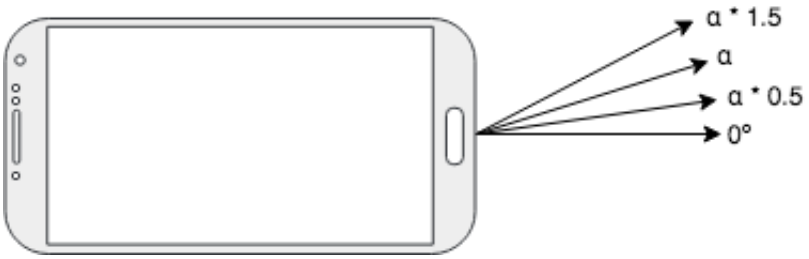


Figure 4.2: *Tilt orientations evaluated to correct potential α estimation due to device sensor errors.*

Pattern-Based Object Identifier

The *Pattern-Based Object Identifier* implements the pattern matching between the skyline extracted from the DEM and the skyline visible in the camera view, and computes *Object Position Corrections* based on the outcome of such procedure. As a result, substantial errors in the DEM, GPS position and orientation sensors are corrected automatically in real-time. This component has been realized starting from the experience for offline peak detection described in Section 2.6.3, introducing significant improvements.

Non-Zero Tilt. The web version of the matching algorithm assumes the camera tilt as negligible (equal to 0) and reduces the problem to the alignment between two cylinders, avoiding the (much more costly) spherical match. This assumption proved viable experimentally; mountain ranges are far from the position of the user and the error induced by a moderate tilt is compensated by the skyline matching algorithm. On a mobile device the assumption of zero or constant tilt must be relaxed, to cope with the movements of the mobile device made by the user during a viewing or shooting session. To avoid switching from 2D cylindrical to 3D spherical alignment, which would jeopardize the response time, we designed an approximate approach: the input image is rotated by the tilt provided by the orientation sensor, standard 2D alignment is performed, and the final peak coordinates are rotated in the inverse direction at the end. Nonetheless, given the noisy nature of the orientation sensors, such tilt estimation is prone to errors. In order to deal with such problem, the input image considers four different tilt orientations, as depicted in Figure 4.2:

- $0°$
- α
- $\alpha * 0.5$
- $\alpha * 1.5$

where α is the tilt orientation estimation output by the device.

Next, the image is rotated into each of the tilt orientations, the alignment is computed and the final tilt orientation to use is chosen based on the best alignment overlap achieved. This method deals with tilting effectively and preserves the fast response time of the 2D alignment, to obtain corrections to the 3D object

positions.

Skyline Detection. The heuristic methods described in Section 2.6.3 work well for offline peak detection, because they are applied to pre-filtered images (fixed webcams have a view that does not change and can be manually checked once and for all for suitability; user generated photos go through an offline binary classification step to retain only samples with obstacle-free skyline view). But they are not well suited to a mobile AR scenario, where it is more likely that the camera is used in adverse weather conditions and in the presence of transient occlusions of the skyline. An erroneous skyline detection can hamper the alignment with the DEM and the positioning of peaks, yielding an unacceptable user's experience. To increase robustness even to small, transient occlusions and find the actual *landscape skyline*, i.e., the set of all points that represent the boundary between terrain slopes and the sky, we have applied a Fully Convolutional Network, as already discussed in Section 3. A pixel-wise model has been embedded within the application to be executed efficiently on-board the devices. More details are included in Section 5 and Section 6.

Occlusion Management. The virtual panorama view contains only the peaks that could be visible by an observer based on the elevation model; in the real image, virtually visible peaks can be occluded by irrelevant objects, such as houses, people or even clouds or fog. The FCN model used for skyline filtering in the mobile AR scenario helps deal with occlusions: the network is trained to recognize the *landscape skyline*, i.e., the portion of the topmost edges that actually represent the boundary of a mountain slope. This capability supports effective occlusion detection. Given a correct alignment between the landscape skyline of the image and the virtual skyline of the panorama, the peaks that are actually visible in the image will have fragments of the landscape skyline in their vicinity, while occluded peaks will not. Thus, once the alignment is found, for each peak a visibility score v is defined as the number of landscape skyline points located no farther than d pixels from the peak position. A peak is considered visible if $v \geq \bar{v}$ (where $\bar{v}$ is a fixed threshold). If a peak is considered visible, its appearance patch is extracted and cached; otherwise no patch is extracted (or its patch is removed from the cache, if previously stored). In this way, the Similarity-Based Peak Identifier will not find the patch inside the future frames.

Sensor Orientation. The sensed orientation of the device can be used to improve the performance of the object identification. Since the match between the virtual panorama and image skyline is approximate, each candidate peak position receives a score, which is an estimate of the confidence of the match algorithm. Such score can be manipulated to take into account the agreement between orientation as sensed from the compass and estimated by the *Position Alignment Manager*. For example, a kernel function based on the difference between the sensed and estimated orientation can be used as a scale factor. Furthermore, the computation of peak alignment can be avoided in the areas of the image in which the kernel factor is equal to zero, because those regions would provide an unre-

liable peak position estimation. Such optimization decreases the computational time: we assume a maximum 50° orientation sensor error and perform the photo-to-panorama alignment not in the whole 360° panorama, but in a $100° + FOV$ portion of it.

4.4.2 Data management

The realization of an outdoor mobile AR application for mountain identification relies on a multi-stage and multi-source data processing pipeline, portrayed in Figure 4.3.

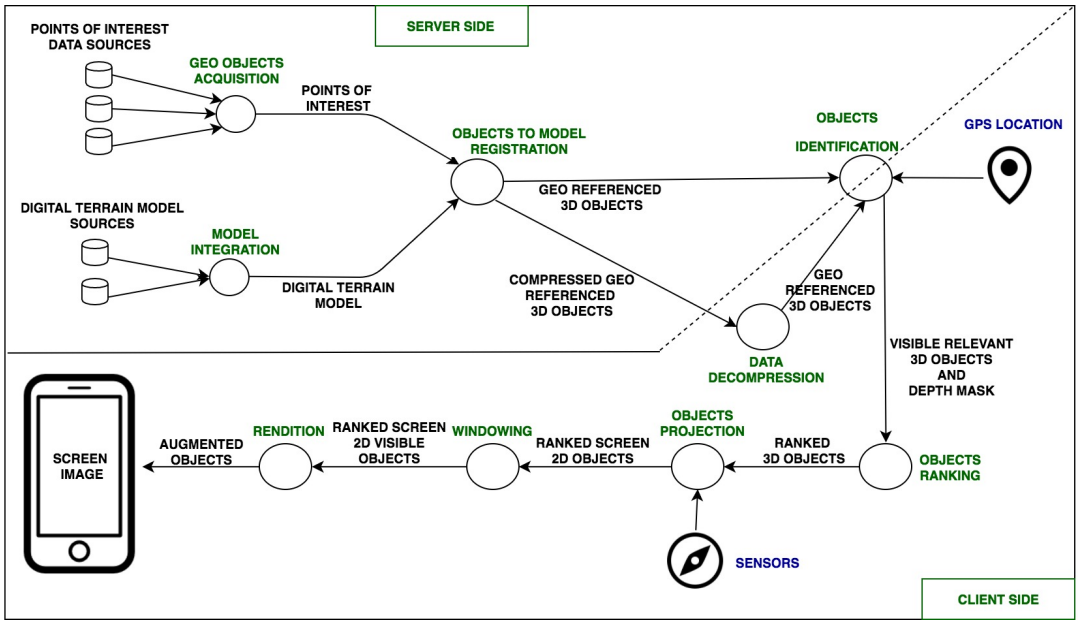


Figure 4.3: *Data management pipelines of an outdoor mobile AR application*

Such pipeline starts with two main processes performed independently at the server side.

- *Geo objects acquisition* addresses the collection and fusion of meta-data about multiple points of interest, possibly from diverse data sources. This step outputs geo-referenced objects and the meta-data about them, which are used to create the augmentation.
- *Model integration* assembles the digital model of the terrain, exploited to create a virtual representation of the panorama the user should be looking at. Also this step may take as input and integrate multiple data sets, e.g. Digital Elevation Models of different parts of the world, with different resolutions. The output is the terrain model integrated with geo-referenced objects of interest, used for recognizing and augmenting the scene viewed by the user.

The *Object to model registration* step packages the geo-referenced objects and the terrain model data to prepare them for subsequent elaboration. If the geo-referenced objects lack altitude information, it also estimates their altitude from the digital terrain model and generates as output the complete 3D coordinates.

This step is performed by default at the server side. However, for the application to work also without Internet connectivity, model and object data can be pre-processed to enable transmission and use at the client side. Such preprocessing requires data compression and segmentation into meaningful units of manageable size (e.g. data can be partitioned based on geographical entities, such as countries or regions).

The *Objects identification* step exploits the GPS location of the user to compute the subset of the geo-referenced objects that are visible from it. It generates from the digital terrain model a 360 degree depth mask of the panorama surrounding the user's position and applies a visibility filter to eliminate the objects that are hidden by the terrain morphology. Figure 4.4 shows an example of virtual panorama generated from the position at coordinates (45.882668,9.22923); the depth mask is represented by the gray scale color of each pixel, which is proportional to the distance from the viewer, and the visible peaks are shown as labels on top of the skyline. This step can be performed at the client side, if the compressed and segmented object and terrain data have been downloaded for offline usage; otherwise, it requires Internet connection to submit the request to the server side. The output is the set of visible 3D geo-referenced objects and the depth mask used to compute them, shown in Figure 4.4.

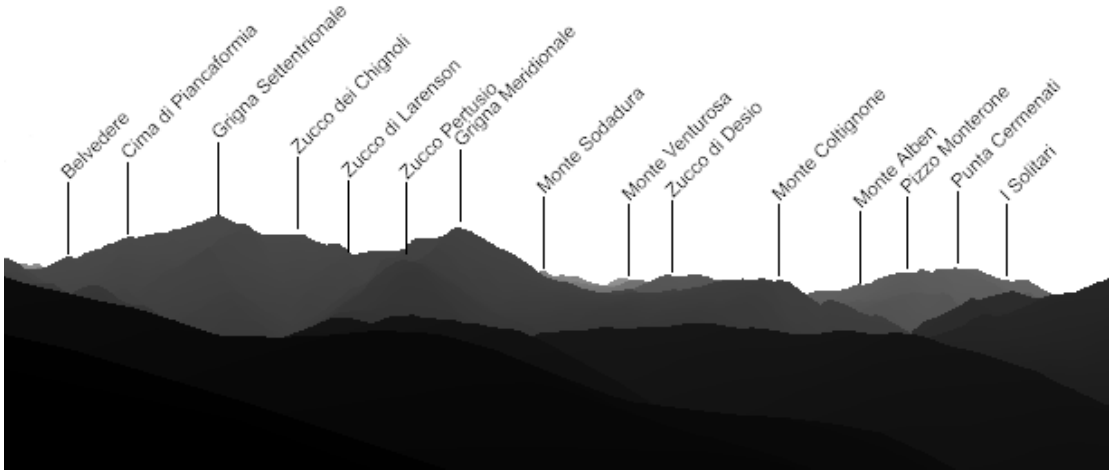
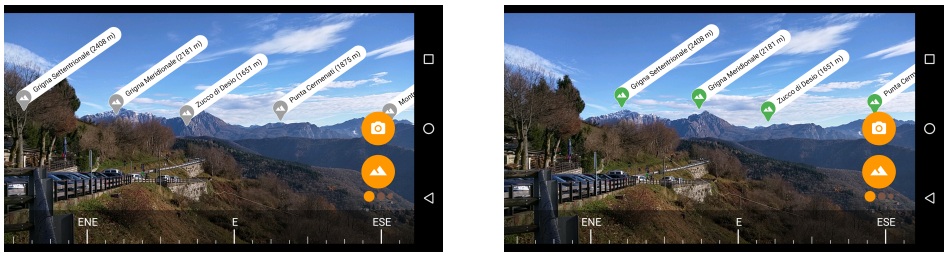


Figure 4.4: *Virtual panorama with depth mask, shown by the gray scale color of pixels, and with visible skyline peaks.*

The *Objects ranking* step takes in input the visible objects and sorts them based on a user-defined criteria (e.g. height, distance, popularity, etc). This step is performed at the client side to avoid server round-trips when the user changes the ranking criterion. The ranking helps choosing the objects to enrich and display, when they are too many to fit the screen.

The *Objects projection* and *Windowing* steps: the former takes into account the current sensor readings (including, e.g. compass, accelerometer and gyroscope), estimates the orientation and field of view of the user, and converts the 3D co-ordinates of objects into 2D screen coordinates. The latter identifies the objects that fall outside the current field of view and should not be displayed. These steps

must be taken at the client side to cope with the fact that the user may move the phone and frame different views from the same position, which requires the real-time re-computation of the on screen coordinates of the visible objects.



(a) *Incorrect sensor-based* **(b)** *Correct CV-based*

Figure 4.5: *(a) presents a sensor-based configuration affected by noise that misplaces the peak labels. (b) presents a CV-based configuration that displays the peak labels in the correct positions.*

Finally, the *Rendering* step actually creates the augmented view, by overlaying the visual representation of the object meta-data (e.g, an icon and/or label) onto the screen image of the device at the proper 2D coordinates. An example of rendition is shown in Figure 4.5, where to highlight the importance of sensor error correction we included (a) incorrect sensor-based rendition based only on GPS and noisy compass sensor readings; and (b) correct CV-based rendering.

Figure 4.6: *An example of the image-to-terrain alignment: input frame (top left), skyline detection (top right), alignment between corresponding panorama and skyline detected (bottom left), final result (bottom right).*

Figure 4.6 shows an example of the sensors error compensation performed by the Object identification step by means of the image-to-terrain alignment and peak position adjustment: the input frame image is captured by the camera of

the device (top left); the skyline from the frame image is extracted (highlighted with green color, top right); the alignment between the virtual panorama extracted from the DEM and the skyline taken from the camera view is executed (bottom left), so as to correct the position of the visible peaks; and peaks are projected onto the screen with the corrected 2D coordinates (bottom right).

4.4.3 Dimensions of heterogeneous augmentation data

Table 4.1 shows a number of examples of data sources potentially relevant for the integration in a mountain mobile outdoor AR application for citizens and tourists, and characterizes them using the above mentioned dimensions.

PeakLens manages a subset of the types of objects mentioned in Table 4.1 (DEM and peaks data), harvests data from multiple sources with reconciliation done offline at the server side, has a cloud-enabled architecture with data caching and compression, displays (at present) only objects that can be rendered as points with text (names with corresponding heights or distances from the user's location), ranks objects by altitude, and provides a pagination and scrolling mechanism to cope with visual clutter.

4.4.4 Usage evaluation

In this section, we summarize the usage experience reported by the users of PeakLens app.

PeakLens was publicly released for Android devices in the Google Play Store[1] in February 2017. Since then, the application usage continues spontaneously and steadily growing in the total number of installations, daily active users, and world distribution of users. So far, the application has been installed by more than 500k users and has received ratings with an average of 3,9/5. Almost half of the ratings include reviews, most of which contained positive feedback. Negative reviews and change requests reflect data quality, management and usage issues and precision issues.

Data quality feedback mostly focuses on either the absence of peak meta-data (because the crawled data sources do not contain it) or on wrong meta-data; the latter case involves either wrong coordinates, which result in the imprecise projection of the peak onto the 2D screen, or on wrong altitude data. To address this issue, we are working on a consensus scheme to fuse conflicting data (e.g. different altitude values) coming from alternate data sources. We are procuring high quality local data sets to estimate the average error of public global data sources and will use a reliability score for deciding the most accurate data to use in the event of conflicts.

Data management feedback mostly refers to data storage: most comments require an option to store offline data in the SD card of the device, due to the limited space in the primary storage. This modification has already been included, enabling the possibility of downloading compressed objects and DEM data segmented at national and regional scale on the SD card of the device.

[1] http://play.google.com/store/apps/details?id=com.peaklens.ar

Semantic	Provenance	Storage/ Availability	Compression	Media type	Visualization
DEM	NASA SRTM	Server, client/ online, offline	Yes	Point cloud	Virtual panorama
	ASTER GDEM	Server, client/ online, offline	Yes	Point cloud	Virtual panorama
Peaks	Peakware	Server, client/ online, offline	No	Text, image, link	Point
	OpenStreetMap	Server, client/ online, offline	No	Text	Point
	PeakBagger	Server, client/ online, offline	No	Text, image, link	Point
Waterbody	Waterways guide	Server, client/ online, offline	Yes	Text, image, link	2D area
	GeoNames	Server, client/ online, offline	Yes	Text, link	2D area
	Wikidata	Server, client/ online, offline	No	Text, link	Point
Alpine huts	Norwegian trekking association	Server, client/ online, offline	No	Text	Point
	Mountainhuts	Server, client/ online, offline	No	Text	Point
Castles	GPS Data Team	Server, client/ online, offline	No	Text	Point
	Wikidata	Server, client/ online, offline	No	Text, image, link	Point
Trails	Norwegian trekking association	Server, client/ online, offline	Yes	Text	Polyline
	Wikiloc	Server, client/ online, offline	Yes	Text, image, link	Polyline
	TrailForks	Server, client/ online, offline	Yes	Text, image, link	Polyline
Towns	OpenStreetMap	Server, client/ online, offline	Yes	Text	2D area
	GeoNames	Server, client/ online, offline	Yes	Text, link	Point
Events	Get Events	Server/ online	No	Text, image, link	2D area
	Facebook	Server/ online	No	Text, link	2D area

Table 4.1: *Examples of data sources and their classifications*

A frequent comment on data usage requires more intuitive object visualization, because several users did not recognize the presence of a scroll button to visualize multiple subsets of the peaks and, thus believed that some of the peaks in view were not identified. Although the design principle pursued so far in PeakLens is to minimize the number of commands, to cope with visualization clutter, we are studying a simple filtering mechanism whereby the user may define the altitude or distance range of the displayed peaks; this mechanism will be backed with proper defaults and used in conjunction with the current pagination mechanism; a self-configurable parameter will also be introduced, which adapts the maximum number of peaks visualized by default to the corresponding screen density.

Precision issues concentrated on the most difficult challenge to face for mountain mobile AR apps: aligning the camera view and the virtual panorama in real-time. This core functionality heavily depends on the reliability of the device's sensors. An error in the GPS position or in the compass orientation produces a virtual panorama that cannot be correctly aligned with the real mountain skyline, resulting in the misplacement of peaks shown in Figure 4.5 (a). However, this capability is also the one for which PeakLens outperforms other applications, which solely depend on the sensor data or on the manual adjustment of the compass orientation. The current version of PeakLens extracts the skyline from the camera view with high precision and speed and can search for an alignment not only in the field of view, but also in a wider virtual panorama, which permits the automatic correction of substantial angular errors in the compass orientation. A forthcoming version of the application will be able to perform the match on the complete 360 degree panorama and thus will also work without the compass sensor or with a low precision one.

CHAPTER 5

Deep Learning model optimization for low-power systems

Convolutional Neural Networks (CNN) have recently exhibited superior performance in a wide variety of applications [59] [85] [121] [173], but their training is a labor intensive task and their execution requires non negligible computational resources. While the general trend was initially orientated towards the achievement of high accuracy by means of defining significantly big and complex architectures that required the computation of billions of floating point operations (FLOPs) [86] [98] [202] [209], such models were not entirely suitable for deployment on mobile platforms with real-time requirements and hardware constraints. This prompted the need to develop more efficient models capable of balancing the trade-off between resources utilization, latency and accuracy.

In this chapter, we report on the experimentation with architectures based on state-of-the-art computation-efficient building blocks in order to obtain high performance, and apply them to optimize the skyline detection models presented in Section 3, thus reducing resources overhead and enhancing their suitability for computationally limited devices.

5.1 Requirements

The main challenge of developing optimized models for DL-enhanced Computer Vision AR modules deployed in low power devices is the need to provide high performance without compromising accuracy and the resulting user experience. These competing objectives can be translated into the following requirements:

- **Minimum parameters.** The model size should be reduced in order to boost the optimization of storage and memory occupation. An acceptable memory

overhead is highly important to fit the model into RAM memory, or ideally cache, thus reducing latency and energy consumption.

- **Optimized computation.** The number of operations associated to the model should be reduced and parallelized as much as possible, so as to accelerate execution, ideally reaching real-time performance. This is directly correlated with the number of parameters, and the type and number of underlying operations.
- **Minimum accuracy loss.** The model should preserve high recognition accuracy in order to provide a satisfying user experience.

5.2 Model optimization

Multi-task learning [84], ensembles [183] and multiple cascaded models [159] are capable of improving robustness and accuracy results w.r.t. single-based models. Such is the case of the combined CNN skyline detection model presented in Section 3.3, which could be further extended by incorporating a mountain classifier to determine whether an image contains a mountain and pre-filter it out if it does not. Nonetheless, said model is meant to be embedded within hardware constrained devices for a use case with real-time requirements and the trade-off between performance and accuracy should be taken into account.

The combined CNN achieves an overall enhancement of ≈10% for ANSA and 0.3% for AA metrics, but requires double computation for each image. Therefore, we have decided to focus on the optimization of the single component for pixel-wise CNN identification, also referred to as *PL Original Model* henceforth. We tackled the reduction of both model size and computational operations, pursuing the acceleration of execution time without being detrimental to the overall accuracy. This iterative process consisted in the proposal and evaluation of over 50 architectures that required the tuning of training hyperparameters, identification of bottlenecks, and modification, addition and removal of entire layers along the architectures. Next, we present the two most efficient, accurate and distinct CNN models produced.

5.2.1 Depthwise Separable Convolutional Neural Network

The first optimized Convolutional Neural Network produced, referred to as *PL Optmized Model v1* henceforth, incorporated Depthwise Separable Convolutions [42] [94] and removed MaxPooling layers by replacing them with Convolutions with stride greater than 1, as also suggested by Li et al. [132]. The filter size of the first standard Convolutional layer from the original architecture has been modified to 3x3, which halves its computational cost and parameters. All Convolutional layers comprise 3x3 filters and the number of filters doubles at each Depthwise Separable Convolution with stride 1, producing at most 128 feature maps in the penultimate layer, previous to the last standard Convolution. This optimized version comprises more Convolutional layers than the original version, but the reduction of kernel dimensions and the introduction of Depthwise Separable Convolutions produce a significant reduction of both parameters and computational

cost. The architecture is presented in Table 5.1: each Depthwise Separable Convolution block consists of a sequence of Depthwise Convolution, Relu activation function, Pointwise Convolution, and Relu activation function. Figure 5.1 depicts the composition of the architecture with the content of said module.

Layer Type	Input Shape	Filter Shape	Stride
Conv	29 x 29 x 3	3 x 3 x 3 x 32	1
ReLU	27 x 27 x 32	-	1
Depthwise Separable Conv	27 x 27 x 32	3 x 3 x 32 x 32	2
Depthwise Separable Conv	13 x 13 x 32	3 x 3 x 32 x 64	1
Depthwise Separable Conv	11 x 11 x 64	3 x 3 x 64 x 64	2
Depthwise Separable Conv	5 x 5 x 64	3 x 3 x 64 x 128	1
Conv	3 x 3 x 128	3 x 3 x 128 x 2	1

Table 5.1: *PL Original Model v1 architecture.*

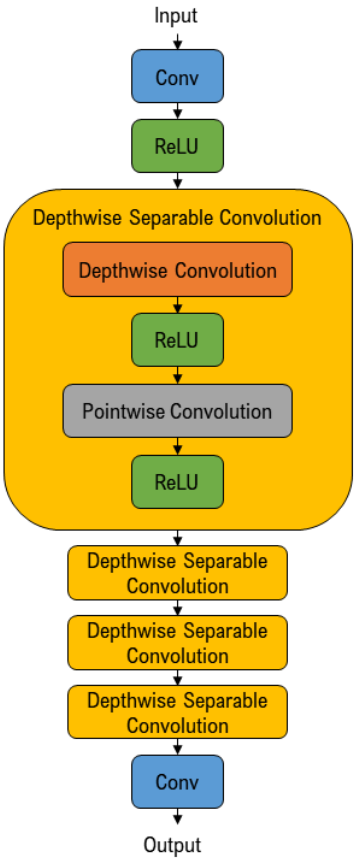


Figure 5.1: *PL Optmized Model v1 architecture overview.*

5.2.2 Inverted Residual with Linear Bottleneck Neural Network

The second efficient Convolutional Neural Network produced, referred to as *PL Optmized Model v2* henceforth, is based on the introduction of Inverted Residual with Linear Bottleneck [191]. The first standard Convolutional layer comprises a

reduced number of filters, but equivalent filter size w.r.t. the PL Original Model. All the remaining layers correspond to Inverted Residual with Linear Bottleneck modules, except for the last standard Convolutional layer. The expansion factor for Inverted Residual with Linear Bottleneck blocks has been set to 3 and each of the modules with stride equal to 2, halves the output height and width w.r.t. its corresponding input volume. The number of layers is increased, but the number of parameters and operations is reduced significantly, requiring at most a filter depth of 32 channels. The architecture is introduced in Table 5.2 and a visual overview of it with the corresponding content of Inverted Residual with Linear Bottleneck modules is presented in Figure 5.2.

Layer Type	Input Shape	Filter Shape	Expansion Factor	Stride
Conv	29 x 29 x 3	6 x 6 x 3 x 16	-	1
Bottleneck	24 x 24 x 16	-	3	2
Bottleneck	12 x 12 x 16	-	3	1
Bottleneck	12 x 12 x 16	-	3	2
Bottleneck	6 x 6 x 24	-	3	1
Bottleneck	6 x 6 x 24	-	3	2
Conv	3 x 3 x 32	3 x 3 x 32 x 2	-	1

Table 5.2: *PL Optmized Model v2 architecture.*

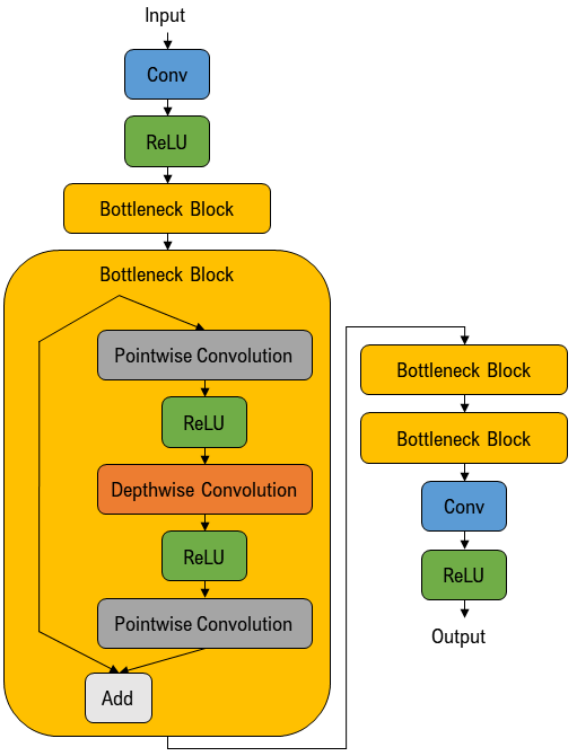


Figure 5.2: *PL Optmized Model v2 architecture overview.*

5.3 Evaluation

The model optimization process has resulted in the construction and training of multiple models, out of which two final implementations were reached. This was possible by iteratively refining such optimized candidate models with the feedback obtained from the evaluation metrics. In this section, we report on the evaluation of the final optimized models in order to assess their value by confronting them w.r.t. the PL Original Model.

5.3.1 Experimental setup

The models were trained by using TensorFlow framework and the evaluation concerning their model characteristics was handled statically and did not require any particular setup.

Moreover, the evaluation required the experimental setup for the assessment of skyline extraction quality, which entailed using the complete test set of 1,771 images, as specified in Section 3.4.1.

5.3.2 Metrics

Next, we introduce the model metrics, which are suited to assess the fulfillment of the requirements defined in this Chapter:

- **Parameters.** The number and representation type of weight parameters comprised by a model defines its overall size and can be calculated statically based on its architecture.
- **Computational cost.** The number of Mult-Adds involved in Convolutional operations is one of the most widely used metrics to measure performance of Deep Learning models. This calculation can also be performed statically, as discussed in [191].
- **Accuracy.** The accuracy of a model is assessed by analyzing its patch-level accuracy, and, most importantly, the overall Average Skyline Accuracy (ASA), Average No Skyline Accuracy (ANSA) and Average Accuracy (AA) quality metrics proposed in Section 3.4.2 by comparing the skyline detected by the model w.r.t. the ground truth.

5.3.3 Experimental results

The underlying number of parameters and Mult-Add operations involved in the models under consideration are reported in Table 5.3. The metrics show that both optimized models are significantly more efficient than the PL Original Model, given that they comprise over 20x less parameters and between 6.7x and 10x less computational cost for 320x240 images. PL Optmized Model v1 contains 20,578 parameters and 198M Mult-Adds, which means one order of magnitude less in the number of operations w.r.t. the PL Original Model. PL Optmized Model v2 contains 18,082 parameters and requires 295M Mult-Adds. Therefore, among the optimized models, the structure of PL Optmized Model v2 is slightly smaller, but PL Optmized Model v1 is more efficient, as its computational cost represents

67.12% of PL Optmized Model v2. Additionally, the large reduction of parameters provides the possibility to potentially 1. fit the models into Cache for more efficient execution. 2. increase the input size of images to process in devices with limited RAM memory, provided that the RAM capacity determines an upper bound and it was already alleviated by the model size reduction.

Model	Parameters	Mult-Adds
PL Original Model	428,732	2G
PL Optmized Model v1	20,578	198M
PL Optmized Model v2	18,082	295M

Table 5.3: *Models size and computational cost.*

The accuracy quality results are reported in Table 5.4 and were calculated with their corresponding optimal thresholds, obtained by assessing maximum accuracy values over the validation set. In general, the optimization of a model tends to entail certain trade-off in terms of accuracy and performance. Nonetheless, the results were very satisfying provided that not only were the model sizes and computational costs lowered by more efficient implementations, but also the final accuracies accomplished by both candidate optimized models were significantly improved. The PL Original Model had already achieved positive overall results, but was not optimized and had some difficulties to deal with non-skyline objects. On the other hand, the optimized models were able to achieve improved accuracy results for all the metrics, especially demonstrating to be better to discard false positive elements from the mountain profiles. This can be justified by the fact that the iterative process of architecture selection with the introduction of more efficient state-of-the-art building blocks and further hyper parameter tuning contributed towards the reduction of overfitting.

Model	Patch Accuracy	ASA	ANSA	AA
PL Original Model	95.81%	89.82%	34.56%	85.42%
PL Optmized Model v1	96.38%	92.09%	48.75%	88.64%
PL Optmized Model v2	96.48%	-%	-%	-%

Table 5.4: *Models accuracy.*

In conclusion, we proposed two additional models that successfully optimized the PL Original Model and demonstrated their effectiveness to improve efficiency and accuracy. Both models were able to perform better than the PL Original Model baseline, but PL Optmized Model v1 is the most suitable solution given its reduced amount of required computational cost and almost equal accuracy w.r.t. PL Optmized Model v2.

5.4 Discussion

5.4.1 Limits to generalization

As also discussed in [191], analyzing the computational cost by calculating the number of Mult-Adds involved in Convolutional operations is an indirect, still

useful metric to assess performance of a model. However, there are additional relevant factors to take into consideration, which determine the resulting execution time, such as the memory access cost, the degree of parallelism of the model and the target platform characteristics. In particular, the on-board inference evaluation on multiple devices is significant to this work given the widely heterogeneous hardware comprised by mobile devices in the market. Inference evaluation on mobile devices with limited hardware is properly assessed in Chapter 6.

CHAPTER 6

Deep Learning inference optimization for low-power systems

Artificial Intelligence on the edge is a matter of great importance towards a better understanding of the world and enhancement of smart devices that rely on operations with real-time constraints. Despite the rapid growth of computational power in embedded systems, such as smartphones, wearable devices, drones and FPGAs, the deployment of highly complex and considerably big models remains challenging. Optimized execution requires managing memory allocation efficiently to avoid overloading, and exploiting the available hardware resources for acceleration, which is not trivial given the non standardized access to such resources.

In this chapter, we present PolimiDL[1], an open-source framework for accelerated DL inference on mobile and embedded systems. PolimiDL speeds-up the execution time of ready-to-use models, by applying multiple optimization methods, and increases efficiency of operations without impacting accuracy. Its implementation is very generic, with neither hardware nor platform specific components, and supports devices with very heterogeneous architectures. The development of PolimiDL was started with the goal of deploying DL models on mobile devices when no other stable solutions were available, and it is currently deployed in PeakLens app.

This chapter includes material from the following publication, co-authored by the candidate: [64].

[1] https://github.com/darianfrajberg/polimidl

6.1 Requirements

Before introducing the architecture and use of PolimiDL, we pinpoint the requirements for its development. When dealing with specific hardware architectures and vendors, maximum performance can be reached by developing ad-hoc optimised solutions. Nonetheless, such approach may comprise scalability and maintenance, when targeting many heterogeneous architectures and devices, as in the case of the Android market nowadays. Moreover, as highlighted in Section 2.3, current acceleration approaches still have limitations: 1. HA primitives are still not completely standardized and stable, but are tightly dependent on SoC vendors; 2. cloud-offloading can imply cost, availability, latency and privacy issues; 3. retraining or modifying the architecture of ready-to-use models can be extremely time-consuming; 4. post-training compression of already small models can detriment accuracy. Under the above mentioned drivers, the requirements at the base of PolimiDL can be summarized as follows:

- **Focus on execution**. It should be possible to train a model using tools already known to the developer. The framework should focus just on execution concerns, without the need of re-training.
- **Minimum dependencies**. It should be possible to execute an optimized model independently from the Operating System, hardware platform or model storage format.
- **Easy embedding**. It should be possible to embed the framework and optimized models into existing applications easily, without the need of ad-hoc integration procedures.
- **End-to-end optimization**. Optimization should be applied as early as possible and span the model life-cycle (generation, compilation, initialization, configuration, execution).
- **Offline support**. Computation should occur only on-board the embedded system, without the need of a network connection for work off-loading.
- **No accuracy loss**. The acceleration for constrained devices should not reduce accuracy w.r.t. the execution on a high performance infrastructure.

6.2 The PolimiDL Framework

PolimiDL aims at speeding-up the execution time of ready-to-use models by applying multiple optimizations that increase the efficiency of operations without modifying the model's output. Its implementation is highly generic, with neither hardware nor platform specific components; this enables performance gains on heterogeneous devices and simplifies maintenance, eliminating the need of targeting different platforms by means of different tools. It is written in C++ and can be compiled for all major platforms, requiring only a very simple interface layer to interact with the platform-specific code. PolimiDL exploits multi-threaded execu-

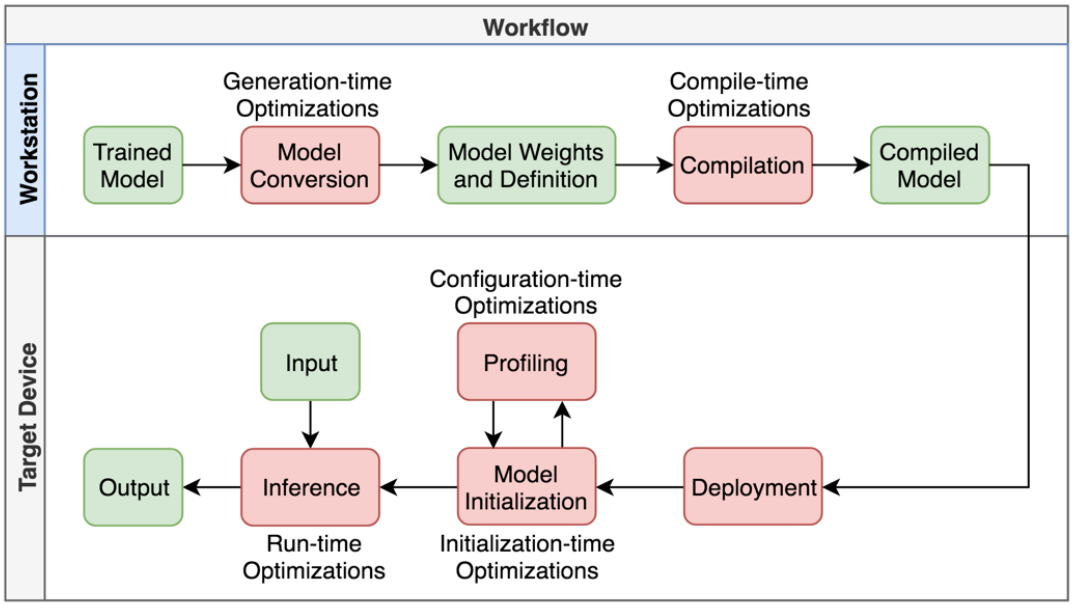


Figure 6.1: *PolimiDL's workflow.*

tion, based on the STL Concurrency Extensions[2], and SIMD instructions, based on the well-known Eigen Library[3].

Figure 6.1 illustrates the general work-flow of the proposed framework, with its main stages (in red) and data/artifacts (in green), showing the stage in which each optimization takes place. The pipeline starts by training a model via some external DL framework, such as TensorFlow or Caffe2, on a workstation or cloud accelerated learning infrastructure, such as Google Cloud[4]. The trained model is converted into a PolimiDL compatible format, while applying generation-time optimizations. Next, the model is compiled for the target architectures and compile-time optimizations are applied, enabling SIMD instructions where supported. Once the model is deployed on the target device, an initialization stage applies initialization-time optimizations to determine the best memory layout. The first time a model is deployed, the initialization step can include the profiling of the model, which enables configuration-time optimizations to determine the best scheduling approach. Finally, the model is ready to process inputs by applying run-time optimizations, which involve dynamic workload scheduling to speed-up inference.

6.2.1 Generation-time optimizations

Layers fusion.

Consecutive in-place layers with identical filter size can be fused into one single layer, thus reducing the number of iterations over the cells of an input matrix. Such technique has been applied to fuse multiple combinations of layers,

[2]https://isocpp.org/wiki/faq/cpp11-library-concurrency
[3]https://eigen.tuxfamily.org
[4]https://cloud.google.com/products/ai/

such as Batch Normalization/ReLu6 and Bias/ReLu. Potentially, Batch Normalization/ReLu6 fusion can be further extended by incorporating a Pointwise Convolution beforehand, taking into account that such combination of layers is frequently used for Depthwise Separable Convolutions.

Weights fusion.

Layers applying functions with constant terms comprising multiple weights can be pre-computed and encoded as unique constant weights, thus reducing the operations at run-time and potentially avoiding temporary memory allocation for such layers. Weight fusion applies, e.g. to the Batch Normalization (BN) layer, in which a subset of the vector weights involved in the normalization, scale and shift steps (γ,σ^2,ϵ) can be factored into a constant vector weight (ω) as follows:

$$BN(x_i) = \gamma * \left(\frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta \tag{6.1}$$

$$\omega = \frac{\gamma}{\sqrt{\sigma^2 + \epsilon}} \tag{6.2}$$

$$BN(x_i) = \omega * (x_i - \mu) + \beta \tag{6.3}$$

where:

- x_i is the input of the layer
- γ, μ, σ^2, β are constant vector weights
- ϵ is a constant value

Weights rearrangement.

Layers' weights, which are multidimensional matrices, are generally stored as linear arrays ordered by a default schema (e.g. output channel, row, column and input channel). PolimiDL stores the weights in different ways based on the layer type. Weights associated to predefined Convolutional layer types are stored in such an order that Eigen's GEMM matrix operations do not require any memory reshaping at run-time. These optimizations are executed automatically and transparently for the developer, who does not need to know their details.

6.2.2 Compile-time optimizations

Fixed network architecture.

The architecture of a model is fixed at compile-time, which enables the compiler to perform per-layer optimizations. As an example, Convolutional layers can exploit loop-unrolling [96], because the number of channels and the kernel size are known at compile-time, potentially generating a different machine code for each configuration of layer parameters. This approach can be seen as a limiting factor, because the model architecture cannot be updated at run-time or by simply changing a configuration file. However, it is important to notice that changing the model architecture is not expected to occur after the model has been deployed.

Besides, in PolimiDL a model can be compiled as a set of Shared Objects (.so) files for the corresponding target architectures (armeabi-v7a, x86 and arm64-v8a for Android), enabling model updates by a simple file replacement. Given a fixed model architecture, PolimiDL supports the update of layer weights at run-time. When the run-time update of weights is not required, then the weights can be stored together with the network architecture, by embedding them in the .so files; this avoids the overhead of loading them from secondary memory, as opposed to TensorFlow Lite, where architecture and weights are stored as an external file loaded from disk.

Shared memory allocation & tick-tock piping.

Efficient memory allocation and management is critical in embedded systems, where the amount of memory is limited and access time is slower than in workstations. Exploiting spatial locality [7] to reduce cache misses can decrease inference time and energy consumption significantly. For this purpose, layers in PolimiDL do not own the memory they read inputs from, write outputs to, or use to store intermediate results: memory is injected as a dependency from the layer scheduler. Given this organization, the memory required by a model can be reduced to just 3 areas: 1. Layer Input 2. Layer Output 3. Temporary data. These areas are properly sized at run-time, to contain the largest layer in the model. A disadvantage of this approach is the need to copy the output of a layer back into the input area to feed it to the next layer. PolimiDL alleviates this inconvenience by inverting the input and output buffers of subsequent layers. With this schema, data goes back and forth between the two buffers in a tick-tock fashion. Tick-tock buffer swapping is skipped for in-place layers, i.e., layers that can use the same buffer area for both input and output: they do not trigger an input/output buffer flip. ReLu layer is a clear example, because it performs value-wide operations enabling in-place modifications. Furthermore, given the fixed model architecture, layer piping can be computed at compile-time via the template meta-programming capabilities of C++, without incurring in any run-time costs.

6.2.3 Initialization-time optimizations

Memory pre-allocation.

Pre-allocating memory buffers to contain the layers of a complete model without memory reuse may be feasible for server computation, but is certainly not the best option for embedded systems with hardware constraints. We have shown how the proposed framework reduces this memory requirements via shared buffers and the tick-tock piping. PolimiDL further reduces memory requirements by fusing the 3 buffers (input, output and temporary) into a single one. During initialization, each layer is queried about its memory requirements: input size, output size and temporary data, which can differ based on hardware capabilities, e.g. number of threads, or input size, in the case of Fully Convolutional Networks. A single buffer is allocated and sized to contain data of the most demanding layer. The upper and lower end of the buffer are used as input/output areas respectively, following the tick-tock strategy, while the area in between is used for the temporary data.

This approach further reduces memory requirements as a single memory cell can store input, output or temporary data in different layers.

Small tasks for low memory consumption.

While some layers require little or no temporary memory to work efficiently, others have a space-time trade-off. As an example, Convolutional layers can exploit SIMD instructions if their 3D input is unrolled into 2D matrices, where each row is the linearized input to the kernel. While unrolling the entire input and exploiting Eigen's SIMD and cache optimization capabilities may reduce the computation time significantly, it also increases the memory requirements of the layer by increasing the size of the temporary buffer. In these cases, PolimiDL does not perform a full input unroll, but divides the operation into smaller tasks, which can be executed independently. In this way, the temporary memory required by the tasks has a fixed size.

6.2.4 Configuration time optimizations

Scheduling optimization.

PolimiDL features a task scheduler, explained in detail in Section 6.2.5, which enables layers to divide the workload into tasks executed by different threads. The optimal size for a scheduled task may vary depending on the specific layer, the underlying architecture, or even on the input size for Fully Convolutional Neural Networks. Task sizes can be considered as parameters, which can be: 1. set to a default value, which may not be optimal 2. inferred by executing a profiling routine during initialization, which may increase the initialization time 3. inferred once for all on the specific device, stored and loaded at subsequent initialization steps. The profiling for each layer is performed by assessing the execution time of different task sizes. A full exploration of the task size space is not possible, given the high time and computation requirements. The sizes used during the assessment are generated by a power law heuristics. Task sizes may be bounded to a maximum value, dictated by the available temporary memory. It is important to notice that the available temporary memory may be more than the one requested at initialization time. This is because the buffer is initialized to contain the largest layer and, as a consequence, layers with smaller footprint can exploit the extra temporary memory.

6.2.5 Run-time optimizations

Dynamic workload scheduling.

Static and even distribution of workload among available cores does not represent the most suitable solution, due to the unpredictable nature of mobile resources availability, more evident in asymmetric architectures such as ARM big.LITTLE [41]. A static scheduling strategy can under-utilize resources, wasting processing power. Conversely, dynamic multi-threaded scheduling of tasks can adapt well to different contexts and allows cores to be better exploited. Tasks are forwarded to a fixed size thread-pool (by default the number of workers is set

to $max(1, \#threads - 1)$). In PolimiDL, during the development of a layer, it is possible to opt-out from dynamic scheduling or to enable it just when profiling shows a significant improvement. Dynamic scheduling should not be applied blindly, as computational intensive layers, such as Convolutions, perform better when dynamically scheduled, while others, such as ReLu, may perform worse due to memory bottlenecks. Therefore, dynamic scheduling is disallowed by default for layers that would be harmed by it.

6.2.6 Layers coverage

Table 6.1 summarizes the layers currently supported by PolimiDL and their features[5].

Layer name	In place	Temporary memory	Schedulable
Convolution	No	Yes	Yes
Depthwise Convolution	No	Yes	Yes
Pointwise Convolution (out channels $\leq$ in channels)	Yes	Yes	Yes
Pointwise Convolution (out channels $>$ in channels)	No	No	Yes
Max Pooling	No	Yes	No
Average Pooling	No	Yes	Yes
Batch Normalization	Yes	No	Yes
Bias	Yes	No	No
ReLu	Yes	No	No
ReLu6	Yes	No	No
Softmax	Yes	No	No

Table 6.1: *Layers supported by PolimiDL.*

Fully Connected layers can be supported by introducing a standard Convolution in which the corresponding kernel size is equal to the expected layer input size. Given an expected input size of 1x1xN, such operation can be managed efficiently by using a 1x1xNxM Pointwise Convolution, where N represents the input channels and M the output classes.

6.3 Evaluation

6.3.1 Experimental setup

The evaluation benchmarks inference execution time of DL models on heterogeneous embedded systems, comparing PolimiDL with the state-of-the-art solution for edge inference: TensorFlow Lite[6]. Measurements are collected by means of an Android benchmark application, implemented by extending TensorFlow Lite's sample application[7] to support multiple experiments. The use of multiple devices

[5]Given the open source release of PolimiDL, the supported layers may be subject to modifications and further extensions.

[6]The latest stable version at the time of writing is tensorflow-lite:1.13.1

[7]https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/examples/android/app

Model	Task	Mult-Adds	Params	Input size
PL Original Model	Segmentation	2G	429K	320x240x3
PL Optmized Model v1	Segmentation	198M	21K	320x240x3
MobileNet	Classification	569M	4.24M	224x224x3

Table 6.2: *Models used for evaluation.*

and models is critical for performance evaluation, given the non-linear correlation between hardware features and tasks characteristics [106].

The evaluation process is conducted as follows:

- Initialization and pre-processing times are not considered in the overall processing time.
- One warm up inference run is executed before the actual measurements.
- 50 consecutive inference iterations are executed and averaged to reduce variance.
- Three complete evaluation sessions with all models and devices are averaged, to further reduce variance.
- Models are run on mobile devices having above 80% of battery charge and pausing for 5 minutes between executions.

Models. Evaluation exploits hardware with limited resources and models with a small-size architecture achieving a good trade-off between accuracy and latency. Three models with diverse characteristics, listed in Table 6.2, are evaluated.

PL Original Model is a Fully Convolutional Neural Network model [140] for the extraction of mountain skylines, which exhibits a good balance between accuracy, memory consumption, and computational cost; it is exploited in the implementation of PeakLens, a real-world AR application for mountain peak recognition on mobile phones. The model was trained with image patches for binary classification, by adapting the LeNet architecture and can be applied to pixel-wise classification of full images. Its architecture is defined in Section 3.3.1.

PL Optmized Model v1 is a modified version of the PL Original Model replacing standard Convolutions with Depthwise Separable Convolutions, inspired by MobileNet [94]. The optimized version improves accuracy and performance and reduces the number of parameters by one order of magnitude. The architecture is defined in Section 5.2.1 and corresponds to the most efficient model for pixel-wise skyline identification obtained.

MobileNet [94] is a well-known state-of-the-art CNN architecture for efficient inference on mobile devices, developed by Google for diverse tasks, such as image classification and object detection. Multiple versions of MobileNet trained on ImageNet are publicly available[8], among which the biggest version has been chosen

[8]https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md

for evaluation (MobileNet_v1_1.0_224).

Devices. Six distinct Android devices with heterogeneous architectures are used, Table 6.3 lists the devices and their characteristics.

Device	Android Version	Chipset	CPU	RAM (GB)
Asus ZenFone 2 ZE500CL (Z00D)	5.0	Z2560 Intel Atom	2-cores 1.6 GHz (4 threads)	2
Google Pixel	9.0	MSM8996 Qualcomm Snapdragon 821	2-cores 2.15 Ghz Kryo + 2-cores 1.6 Ghz Kryo (4 threads)	4
LG G5 SE	7.0	MSM8976 Qualcomm Snapdragon 652	4-cores 1.8 GHz Cortex-A72 + 4-cores 1.2 GHz Cortex-A53 (8 threads)	3
LG Nexus 5X	8.1	MSM8992 Qualcomm Snapdragon 808	4-cores 1.44 GHz Cortex-A53 + 2-cores 1.82 GHz Cortex-A57 (6 threads)	2
Motorola Nexus 6	7.0	Qualcomm Snapdragon 805	4-cores 2.7 GHz Krait (4 threads)	3
One Plus 6T	9.0	SDM845 Qualcomm Snapdragon 845	4x 2.8 GHz Kryo 385 + 4x 1.8 GHz Kryo 385 (8 threads)	6

Table 6.3: *Devices used for evaluation.*

Configurations. Multiple configurations are tested to analyze the impact of the scheduler thread pool size. *#threads* is the number of usable threads, which depends on the device (see Table 6.3). The evaluated configurations comprise:

- **min(4,#threads):** the thread-pool has a maximum of 4 workers, which is TensorFlow Lite's default configuration.
- **max(1,#threads-1):** the thread-pool employs all available threads but one.
- **#threads:** the thread-pool comprises all threads, for maximum parallelism.

6.3.2 Metrics

The main metric taken into consideration for the evaluation of inference is **Latency**, which determines how fast a frame can be processed by measuring the time interval required between input reception and response delivery.

6.3.3 Experimental results

We report the experimental results obtained with TensorFlow Lite and PolimiDL for each combination of model, device, and configuration. Positive results for PolimiDL are highlighted in green in the tables and negative results in red. This Section presents the summarized evaluation results with averaged values and more detailed information to support the stability of the results can be found in Appendix A.

Table 6.4 reports the results for *PL Original Model*. PolimiDL outperforms TensorFlow Lite in almost all cases (highlighted in green), with reductions of up to 57.32% (Motorola Nexus 6); TensorFlow Lite performs better (highlighted in red) just in one device (LG Nexus 5X) with a single configuration. Overall, PolimiDL consistently reduces average execution time by above 30%.

	TensorFlow Lite (ms)			PolimiDL (ms)		
Device	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**
Asus ZenFone 2	1352.67	1672.67	1353.00	936.00 (-30.80%)	1138.00 (-31.96%)	936.67 (-30.77%)
Google Pixel	207.67	255.33	210.33	145.00 (-30.18%)	171.00 (-33.03%)	145.00 (-31.06%)
LG G5 SE	418.67	290.00	272.67	273.00 (-34.79%)	209.00 (-27.93%)	200.33 (-26.53%)
LG Nexus 5X	423.67	370.33	336.33	432.33 (+2.05%)	342.33 (-7.56%)	282.33 (-16.06%)
Motorola Nexus 6	336.67	505.33	337.67	169.00 (-49.80%)	215.67 (-57.32%)	168.33 (-50.15%)
One Plus 6T	176.00	144.33	145.33	104.00 (-40.91%)	91.00 (-36.95%)	89.00 (-38.76%)
			Average	(-30.74%)	(-32.46%)	(-32.22%)

Table 6.4: *Experimental results of PL Original Model.*

Table 6.5 reports the results for *PL Optmized Model v1*. This model is smaller, yet more accurate than the original one. PolimiDL outperforms TensorFlow Lite and reduces inference time significantly. This is due to the design of memory management, which exploits spatial locality well and reduces cache misses. The performance gain is highly consistent: execution times are reduced on average more than 62% in all the configurations. Improvement is particularly sensible for low-end devices, such as ZenFone 2, where the reduction is greater than 77%. This represents a huge enhancement towards the possibility of using low-end devices in use cases that require near real-time processing.

	TensorFlow Lite (ms)			PolimiDL (ms)		
Device	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**
Asus ZenFone 2	740.67	807.67	743.33	166.00 (-77.59%)	179.33 (-77.80%)	167.67 (-77.44%)
Google Pixel	82.00	95.00	82.67	30.00 (-63.41%)	35.33 (-62.81%)	31.00 (-62.50%)
LG G5 SE	185.67	138.33	138.00	94.33 (-49.19%)	68.00 (-50.84%)	70.67 (-48.79%)
LG Nexus 5X	204.33	193.00	181.00	84.67 (-58.56%)	80.33 (-58.38%)	77.00 (-57.46%)
Motorola Nexus 6	140.33	225.67	135.67	52.33 (-62.71%)	66.00 (-70.75%)	49.00 (-63.88%)
One Plus 6T	66.67	68.67	66.33	22.00 (-67.00%)	22.67 (-66.99%)	22.33 (-66.33%)
			Average	(-63.08%)	(-64.59%)	(-62.73%)

Table 6.5: *Experimental results of PL Optmized Model v1.*

Finally, Table 6.6 reports the results for *MobileNet*. Performance of the two

frameworks are quite comparable, but PolimiDL reduces overall execution time. The most significant gains are achieved on the ZenFone 2 and Nexus 6 devices with improvements of up to 51.33% and 41.01% respectively. TensorFlow Lite performs slightly better (not over 5%) on certain settings involving devices with big.LITTLE architecture (LG G5 SE and LG Nexus 5X). Despite the fact that PolimiDL features dynamic scheduling, it is the Operating System the ultimate responsible of the allocation of tasks to workers and low frequency cores seem to be prioritized for this model and devices. Nonetheless, the average execution time, when using all threads but one, is reduced by 17.05%.

	TensorFlow Lite (ms)			**PolimiDL (ms)**		
Device	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**	**Min (4,Threads)**	**Max (1,Threads-1)**	**All Threads**
Asus ZenFone 2	734.00	775.33	733.33	371.00 (-49.46%)	377.33 (-51.33%)	374.33 (-48.95%)
Google Pixel	75.67	82.33	77.00	74.00 (-2.20%)	82.67 (+0.40%)	73.67 (-4.33%)
LG G5 SE	263.67	274.67	275.67	276.67 (+4.93%)	259.00 (-5.70%)	256.33 (-7.01%)
LG Nexus 5X	217.33	225.00	223.33	222.33 (+2.30%)	234.33 (+4.15%)	226.00 (+1.19%)
Motorola Nexus 6	224.33	298.33	227.67	203.67 (-9.21%)	176.00 (-41.01%)	163.33 (-28.26%)
One Plus 6T	56.67	56.67	57.67	49.67 (-12.35%)	51.67 (-8.82%)	53.00 (-8.09%)
			Average	(-11.00%)	(-17.05%)	(-15.91%)

Table 6.6: *Experimental results of MobileNet model.*

The activation of NNAPI has been assessed in TensorFlow Lite for the supported devices, but results are not reported due to unstable performance. NNAPI reduces execution time on the Google Pixel, but doubles it on the LG Nexus 5X.

The observed results show that the size of the model represents the most important factor to maximize the performance in favor of PolimiDL. The smaller the number of parameters of a model, the higher performance improvement is achieved by PolimiDL w.r.t. TensorFlow Lite. Consequently, *PL Optimized Model v1* is the most benefited model, and *PL Original Model* reaches a higher performance improvement w.r.t. *MobileNet*, even though the former comprises a higher number of Mult-Add operations. This is explained by the fact that multiple underlying optimizations present along PolimiDL's workflow, maximize the spatial locality when managing the memory and small models can potentially fit in cache memory, thus accelerating the final execution time.

In conclusion, experimental results demonstrate the potential of PolimiDL by showing competitive results w.r.t. the well-known TensorFlow Lite platform. Results are particularly improved when dealing with small models and low-power devices; this finding corroborates the potential of the proposed framework for supporting the implementation of AR applications for mass market mobile phones, which is the use exemplified by PeakLens app.

6.4 Discussion

6.4.1 Limits to generalization

The applicability of PolimiDL to a specific model is subject to the support of the required layers and to the availability of a converter from the source DL framework format. PolimiDL currently supports the layers included in Table 6.1 and conversion from the TensorFlow format. Simple missing layers, such as Deconvolution [140], Dilated Convolution [37], Leaky ReLU [229] and Global Average Pooling [136], can be easily incorporated into the framework. Furthermore, the generic and extensible architecture of the PolimiDL Converter[9] makes it possible to support the conversion from additional frameworks by simply extending a few abstract classes and methods, to load models and parse their corresponding layers definition and weights. On the other hand, PolimiDL targets devices with limited resources and is currently able to support only simple and small feed-forward DL architectures. Features such as batch inference, model quantization and the inclusion of certain additional layers may require adaptations of the architecture design; for example, the support of state-of-the-art architectures with multiple parallel branches [209] or residual skip connections [184] would require more complex buffers piping, to cope with synchronization and additional memory management. Shared object libraries with self-contained weights declared as variables can be used for small models, common in embedded systems; but they may suffer from compilation constraints when big models, such as VGG-16 [202], are involved. Finally, PolimiDL currently runs on CPU only and does not support GPU processing, due to the still limited and non-standard access to it, which would require multiple implementations.

[9] https://github.com/darianfrajberg/polimidl_converter

CHAPTER 7

Multi-Sensor Mobile Application Testing Framework

Outdoor mobile applications rely on the input of multiple, possibly noisy sensors, such as the camera, GPS, compass, accelerometer and gyroscope. Testing such applications requires the reproduction of the real conditions in which the application works, which are hard to recreate without automated support. This is particularly challenging when CV is used, due to the necessity of recreating the relationship between camera frames and other sensor readings that help infer orientation, motion and view.

This chapter presents a capture & replay framework that automates the testing of mobile outdoor applications; the framework records in real-time data streams from multiple sensors acquired in field conditions, stores them, and enables developers to replay recorded test sequences in lab conditions, also computing quality metrics that help tracing soft errors.

This chapter includes material from the following publication, co-authored by the candidate: [67].

7.1 Requirements

Testing an outdoor AR application is a complex task that requires simultaneously evaluating the precision of object positioning and the response time, two competing objectives, in a realistic setting that considers the sensor inputs (not available in the lab). The assessment criteria must also take into account usage conditions: if the user keeps the device steady, low error is the prominent goal, while higher execution time due to re-positioning after micro-movements is less relevant; conversely, if the device is subject to movement (e.g, during walking), fast execution can be more important than object positioning precision. There-

fore, testing should be supported by an auxiliary architecture that helps achieve the following objectives:

- **Realistic and controlled orchestration.** It should cope with the controlled orchestration of heterogeneous input sensors, while being executed in lab conditions with equivalent behaviour to the real outdoor usage.
- **Performance evaluation.** It should use the performance metrics best suited to a specific application and operating condition.
- **Automated heterogeneous assessment.** It should be performed in an automated manner, so as to facilitate the contrast of different designs in the same operating conditions and assess the same designs under different operating conditions.

Outdoor mobile applications support the activity of users in field conditions, where the task at hand requires the processing of inputs from multiple sensors. The distinctive characteristics of such applications are their dependency on multiple, heterogeneous, and often noisy sensors, in addition to the need to process sensor data streams in real-time to deliver a proper user experience.

Testing an outdoor mobile application requires verifying its behavior, in terms of failures, soft errors, or performance, in *working conditions*. Such working conditions, also called *context* in [179], [218] and [134] comprise the input values of all the sensors in which the application relies on for its functioning. Building a test set that reproduces working conditions faithfully is challenging because most sensors are extremely noisy and their accuracy varies greatly [24], not only on different mobile devices, but also on the same device in different operating conditions (e.g. GPS positioning can be affected by meteorological conditions, compass orientation by the proximity of an electrical source). Furthermore, it is also necessary to take into account the temporal correlation of multiple sensor data streams; for example, in an application that overlays information on the screen based on what the user is looking at, the usage context is composed of the sequence of positions from the GPS sensor, the sequence of orientations of the device from the compass sensor, the sequence of pitch and roll values of the device from the accelerometer and the gyroscope, and the sequence of view frames from the camera. Such sequences are correlated, because the content of the camera frame at a given time depends on the position, orientation, pitch and roll data. In particular, the development of Mobile Augmented Reality (MAR) applications with sophisticated Computer Vision modules, such as [152], [192], [60] and [211], exploit many of the previously mentioned sensors and may benefit from a mechanism to assess their performance realistically.

The goal of testing can be the identification of the insurgence of hard errors, which cause the application to fail, the quantification of performance properties, or the verification of soft errors, i.e., the occurrence of bugs that do not cause the application to fail, but nonetheless degrade its behavior w.r.t. some desirable characteristic that affects user's acceptance. Investigating soft errors requires defining the property to observe, formalizing quality metrics for its evaluation, extracting the values of the target property from application runs, and comparing

the extracted values with some reference, which acts as a *gold standard* (i.e., a representation of what is ideal for the user).

7.2 The Testing Framework

Figure 7.1 portraits the process of testing for soft errors a mobile application that relies on multiple sensors. The *Acquisition and correlation* step collects input data from multiple data sources and manages their dependencies and temporal correlation; it outputs a *test sequence*, which is a temporal series of values, one per type of input.

The *Feeding* step prepares the ground for executing a test run of the application on a test sequence. It encodes the test sequence in the format required by the execution environment and submits it for processing.

The *Execution* step actually runs the test session by executing the application with the test sequence as input. In the case of a mobile application, the execution can be performed on board the device or in a simulator. The *Property extraction* step observes an execution run to fetch the values of the property under examination. This can be normal termination if the testing goal is to uncover failures; resource usage or execution time if the testing goal is to analyze performance; or an application dependent property if soft errors are the target.

The *Evaluation* step concludes the process by reporting the outcome of execution runs. To assess performance and soft errors, the evaluation must characterize the (un)desired behavior by metrics. Such metrics can be the deviation of a directly observable variable from a target value (e.g. the response time exceeding a threshold) or may require comparing some output of the application with an example providing a quality bound (e.g. evaluating the error in tracking the user's location during motion can be done by comparison with a correct sequence of positions; evaluating the misplacement of information over objects on the screen can be done by comparison with a correct sequence of 2D screen object coordinates).

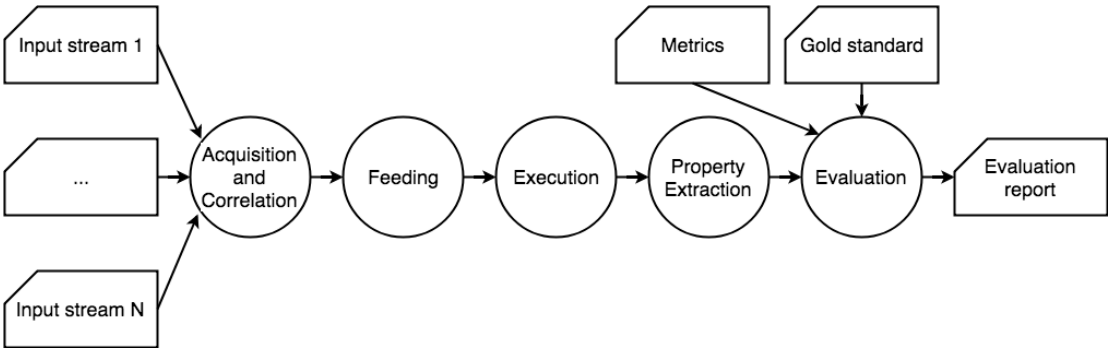


Figure 7.1: *Testing process of a multi-sensor mobile application.*

The implementation of the testing process of Figure 7.1 poses several challenges. The creation of test sequences in the *Acquisition and correlation* step must cope with the heterogeneity and dependencies of input data. Albeit model-driven data generators and databases of traces exist for several classes of sensors (e.g. network connectivity [31], position [213], motion [9]). The construction of

multi-sensor test sequences by means of the temporal sampling of independent data streams for the different sensors is inadequate for testing applications that have interdependency of input values and for evaluating properties for which sensors interdependency cannot be ignored; for example, assessing the presence of soft errors in the screen position of geo-referenced information during the user's motion requires considering the interdependency between camera content and compass position, orientation, pitch and roll. In such a situation, a *multi-sensor data capture* approach, enabling the simultaneous recording of sensor values in field conditions, may be the only viable solution to obtain realistic test sequences and correctly reproduce the usage context for testing purposes. However, the cost of building a multi-sensor data capture tool may be nearly equivalent to that of building the application itself.

The *Feed* step must be able to supply the *Execution* with the test sequence in a way that faithfully reflects the reading of sensor in field conditions. If execution is performed on the device, this requires interfacing the component that implements the *Feed* step to the sensor management services of the mobile operating system; if execution is emulated, the challenge is ensuring that the emulator can be made to supply values to the application at the same rate that would be experienced in the real device [179].

The implementation of the *Property extraction* step distinguishes the case in which the observed property can be computed without access to the internal structure of the application and the case in which such access is required. Whereas failures and performance issues can be detected without access to the source code, soft errors, being application dependent, may not be observable unless the source code is instrumented to export the application status from which the target property can be observed and the metrics computed.

Finally, the challenge of *Evaluation* is the encoding of the testing goal into a computable metrics, whose evaluation may require the construction of a gold standard. The gold standard is a mapping between each element of the test sequence and the corresponding value of the observed property that represents a correct or user acceptable output, given that context. The creation of the gold standard is typically a manual procedure, either because it involves human judgment (deciding what is acceptable) or because an algorithmic solution would have the same complexity (and potentially suffer from the same defects) of the system under test.

7.2.1 Architecture

The proposed architecture relies on a capture and replay framework, which enables the collection of correlated multi-sensor traces in field conditions and produces test sequences that can be used for the controlled execution of the system under test both in the mobile device and in an emulator. Figure 7.2 shows the general system organization.

The **Capture Module** executes in the mobile device and orchestrates the acquisition of multiple sensor data streams. It interfaces to the sensor Application Programming Interface (API) of the mobile device: the Data Acquisition Manager sub-module handles the parallel execution of the data acquisition threads, one per

sensor, and the buffering of the sensed values. The Synchronization Manager is responsible for the temporal alignment of the sensor readings. One sensor is registered as the master, and its callback determines the synchronous reading of the other ones from the buffers. This approach takes into account the fact that the camera sensor is normally the bottleneck in sensor data acquisition. If the camera sensor is registered as the master, the acquisition of each camera frame triggers the reading of the remaining sensor values from the buffers at the time of the callback. The Storage Manager formats the multi-sensor readings in the form of a *test sequence*, encoded in JavaScript Object Notation (Json) format and archived on the local storage of the device. It is worth to mention that the Capture module is executed independently and not in parallel with the Application, as otherwise the outcome would be a lower performance for both of them.

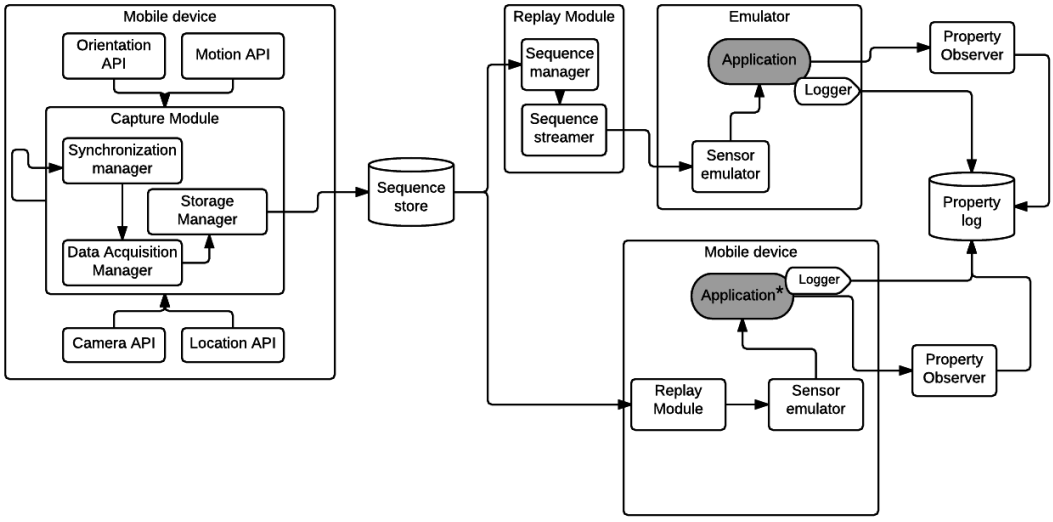


Figure 7.2: *Architecture of the capture and replay framework.*

The **Replay Module** can be executed in the mobile device and in a workstation, in conjunction with a mobile emulator. It comprises a Sequence Manager, which de-serializes an input test sequence into main memory, and a Sequence Streamer, which feeds the sensor data to the (emulated) sensor API of the execution environment. The Sequence Streamer runs in a single thread and handles the feeding of multiple sensor values; it synchronizes on the timestamp of the master (i.e., the slowest) sensor: it fetches the next master sensor reading, gets the correlated values of the other sensors and submits them to the execution environment. The submission rate of the Sequence Streamer is dictated by the acquisition timestamps recorded in the field by the Capture Module. To reproduce the context as faithfully as possible, the Sequence Streamer replays the sensor data series as it is, i.e., without checking the ready status of the application. This mimics the fact that in slow devices the processing rate of sensor values (typically the rate at which camera frames can be analyzed) may be lower than the acquisition rate; this causes the loss of some sensor readings during the live conditions, a situation that must also be reproduced in the testing session.

As usual in context-based approaches that include the reproduction of sensed values, the testing environment must support the replacement of the real sensor APIs with mocked-up interfaces that can serve predetermined data. In the case of emulated execution, the emulator makes the supply of archived sensor data transparent to the application. Conversely, execution in the mobile device requires the installation of a sensor emulation library, which exposes its own interface. Therefore, the execution within the testing environment requires an alternate version of the application, in which the native sensor API calls are replaced with calls to the emulated APIs.

Figure 7.2 also shows the components for extracting the properties necessary for the assessment of soft errors from the test sessions (Logger and Property Observer). These modules are application-specific and are discussed in the next section.

7.2.2 Implementation

The architecture of Figure 7.2 was implemented in Java and supports the testing of Android applications (version 4.0 Ice Cream Sandwich and above). The Capture Module has been interfaced with the following sensor APIs: the Android Camera API[1], the Google Location Services [2], and the Android Sensor API[3]. The latter provides callbacks for different sensors, including gyroscope, accelerometer, and compass. Such information can be interpolated in order to obtain the resulting rotation matrix and orientation vector.

The Storage Manager serializes sensor data into test sequences represented in the Json format illustrated by the following fragment:

```
{"imageName":"20170430_115643_b52b96d9_1.jpg", "rotation":1,
↪ "sensorAccuracy":3, "orVector":"[-2.373061, -0.20468707,
↪ 3.1223032]", "rotMatrix":"[0.15508807, 0.71608853, 0.6805881,
↪ 0.0, 0.13268146, -0.6978047, 0.70394254, 0.0, 0.97896814,
↪ -0.018886, -0.20326078, 0.0, 0.0, 0.0, 0.0, 1.0]",
↪ "timestamp":1493546203647}
```

The values of the camera sensor (frames) are stored externally as files, so that the captured frames can be reused more easily for other purposes (e.g. to build the gold standard data set, see Section 7.3).

The Sequence Streamer of the Replay Module can be interfaced with the Android Studio Emulator. For execution in the testing environment, the alternate version of the application under test must replace the calls to the Android native APIs with calls to the correspondent emulation library APIs.

The addition of another sensor API requires the following steps: 1) the implementation of a SensorEventListener class that listens to the changes in the sensor, computes the values and notifies this event to the application; 2) the registration of the new SensorEventListener to the Data Acquisition Manager of the Capture

[1] https://developer.android.com/guide/topics/media/camera.html [accessed 10 April 2019]

[2] https://developers.google.com/android/reference/com/google/android/gms/location/LocationServices [accessed 10 April 2019]

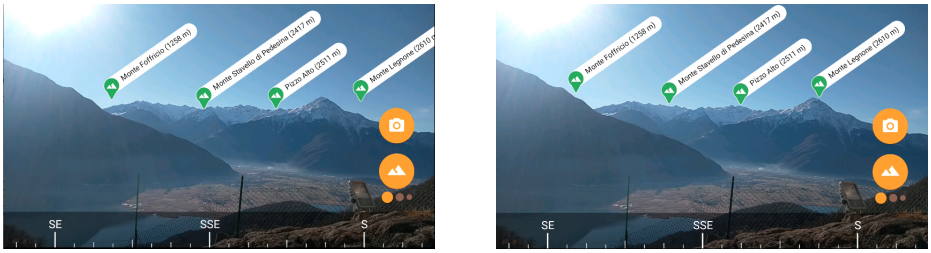
[3] https://developers.android.com/guide/topics/sensors/sensors_overview.html [accessed 10 April 2019]

Module; 3) the addition of the sensor value representation in Json format of the test sequence; 4) the implementation of a sensor play-out class and its registration in the Sequence Streamer of the Replay Module.

7.3 Evaluation

7.3.1 Case study

The multi-sensor testing framework has been applied to PeakLens use case. Such application is a good experimental case study provided that it is designed to work in outdoor conditions and depends on multiple noisy sensors; it acquires the user's location from the GPS, the orientation of the device from the compass sensor, the motion of the device from the gyroscope and accelerometer, and the current view from the camera frames. It analyzes the incoming camera frames with a Computer Vision module, detects the mountain skyline, marks the peaks visible on the skyline with an icon, and labels each identified peak with relevant metadata (name, altitude and/or distance from the viewer).



(a) *Incorrect* **(b)** *Correct*

Figure 7.3: *(a) presents a soft error that causes peak labels to be misplaced with an horizontal offset w.r.t. the correct screen coordinates. (b) presents the peak labels correctly displayed.*

The essential factor that defines the quality of the users experience is the accuracy of labeling the peaks framed by the camera. As Figure 7.3 (a) shows, an error in the computation of the screen coordinates of one or more peaks deeply compromises the utility of the application, as clearly revealed by the user's reviews. In the ideal situation (Figure 7.3 (b)), the application must be able to precisely identify the screen coordinates of the mountain summits that appear in the framed scene and visualize the meta-data in the correct places. Soft errors in the computation of the coordinates of peaks can be revealed by comparing the screens produced by the application with a sequence of artificial screens created by a user who manually labels images, as explained in Section 7.3.2. Such a set of manually annotated images constitutes a gold standard, which can be used to compute the value of metrics that quantify the quality of the user's experience, as explained in Section 7.3.3.

In the reported case study, the illustrated testing process and framework are characterized by the following aspects:

- A test sequence for the application consists of a multi-sensor temporal series, comprising the correlated values of the GPS, compass, accelerometer, gyroscope (if available) and camera sensors.

- Test sequence acquisition and correlation are performed by a *Capture Module*, implemented in the mobile device, which records test sequences in field conditions.
- The feeding of the test sequence is implemented by a *Replay Module*, which services (replays) the elements of a test sequence reproducing the temporal layout and data correlation captured in the field.
- Application execution can be performed both on the mobile device and on an emulator (in the case study, the Android device is used).
- The extracted property for evaluating soft errors is the mountain peak position; a peak position is defined as the pair of 2D screen coordinates of the camera frame at which the summit of a mountain appears and is used to label the peak (as visible in Figure 2.7). Such property can be computed in two ways: by instrumenting the source code or without instrumentation, by capturing and analyzing the application's screen.
- Evaluation is performed by means of metrics that compare the peak positions extracted from the application and the "correct" peak positions. The metrics employ a gold standard data set created with a crowdsourcing system that allows crowd workers to manually specify the position of visible peaks in a series of mountain images.

7.3.2 Experimental setup

The gold standard for assessing the application is defined as a sequence of camera frames, in which each frame portraits an outdoor scene with a mountain skyline and is associated with the set of 2D screen coordinates of (some of) the visible peaks on the skyline.

An effective way to build such a gold standard sequence is to employ the same Capture Module that is used to record the multi-sensor test sequences; from such a sequence it is possible to extract the individual camera frames, and manually annotate them with the 2D coordinates of visible peaks. In the case study, the Capture Module has been employed by a panel of beta testers to gather sequences in diverse mountainous areas around the world.

To support the manual annotation of the peak coordinates in the camera frames, the crowdsourcing Web application (called Peak Annotator) shown in Figure 7.4 has been created.

This Web interface allows a crowd worker to upload a new sequence of multiple frames or to annotate an already existing one. When a sequence is opened, its first frame is displayed, as shown in Figure 7.4. To accelerate the work and allow also non experts to annotate frames, a suggestion about which peaks should be visible is computed and displayed. To this end, the DEM is queried with the position and orientation extracted from the test sequence, a virtual panorama with the candidate visible peaks is displayed in the background of the current frame, as illustrated in Figure 7.4; the worker can simply drag and drop the suggested peak icons to position them in the correct place in the camera frame. Since the default size of a captured sequence is 500 frames, the manual creation of the gold

sequences with the Peak Annotator may be a labor-intensive task. To speed-up the process, the Peak Annotator contains a *Next* button, which shows the successive frame in the sequence with the peaks already pre-positioned on the skyline. The pre-positioning of peaks is performed by exploiting the screen coordinates of the preceding frame and applying a correction based on the projection of the current sensor orientation. Since the frames in the sequence are recorded at a high rate (typically close to 30 per second) and device movements during the capture are slow and continuous, such a simple peak pre-positioning procedure is extremely effective in placing peaks by default; with this simple technique, the number of drag and drop interactions needed to annotate a full sequence is dramatically reduced: down to less than 10% of the peak positions need to be corrected after annotating the first frame. As a further aid to evaluate the quality of the gold standard, the Peak Annotator contains a *Play* button, whereby the user can play out the annotate sequence.

In the case study, 56 sequences were captured and manually annotated, comprising from 100 to 500 frames. They were taken with different devices, under very diverse conditions and locations around the world, and comprised some extremely challenging scenarios. The annotation time of a sequence ranges from less than 5 minutes to around 25 minutes, the longest time being necessary for 500 frames sequences with a lot of fast and irregular device motion during the capture. Afterwards, a cross-validation task to verify the correctness of the annotations was performed, preserving as a result 50 correctly annotated sequences.

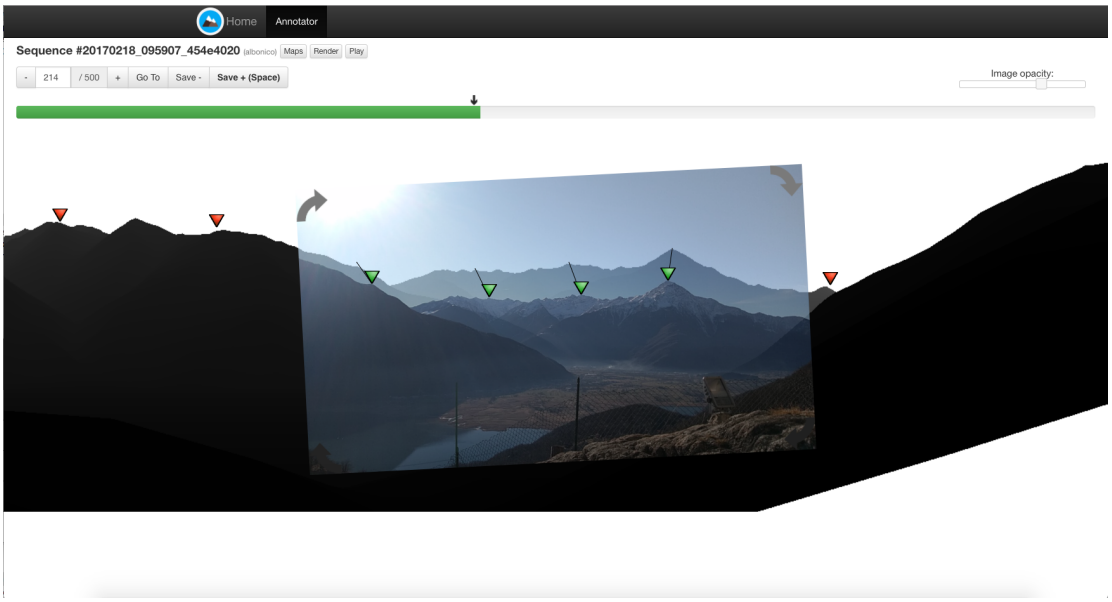


Figure 7.4: *Crowdsourcing user interface for manually annotating the positions of peaks in a sequence of frames. The user can: 1) drag into the correct position (shown in green) candidate peaks suggested by the system; 2) mark candidate peaks as non-visible in the frame (shown in red).*

Furthermore, it is worth to mention the fact that in order to apply the presented testing framework to other use cases, they would require to instantiate their own customized tool for the construction of the corresponding gold standard. The

gold standard definition highly depends on the underlying problem and can not be completely abstracted and generalized. Nonetheless, the components of the interface developed for PeakLens can be taken as baseline and further adapted for other applications without considerable effort.

7.3.3 Metrics

In complex multi-sensor outdoor applications, the success of the application depends primarily on non-functional features such as the *accuracy* of the outputs, while other functions, such as the user interface, storage and network connectivity management are comparatively simpler to implement and converge to stability more easily. In the case study, the following metrics have been defined to quantify the defects in peak positioning that may lower the accuracy of the application. Most of them are rather generic and their application may be suitable for other use cases focused on the augmentation of other elements of interest instead of mountain peaks.

The **Accuracy** measures the fraction of peaks correctly handled, which takes into consideration both visible peaks in the gold standard that are projected in the frame and not visible peaks in the gold standard that are not projected in the frame. The sequence accuracy is the average of its frames.

The **Precision** indicator measures the fraction of peaks positioned in a certain frame of a sequence that are relevant (i.e., appeared also in the same frame of the gold sequence). The overall sequence precision is the average of the precision of every frame. It measures the quantity of false peak positions generated by the application.

The **Recall** metric measures the fraction of peaks present in the frame of the gold sequence that also appears in the corresponding frame of the tested sequence. It evaluates the erroneous omission of peaks from a frame in which they should appear[4]. The sequence recall is the average of its frames.

The **Average Angular Error (AAE)** metric quantifies the positioning errors of all the peaks w.r.t. the position in the gold sequence. Given a frame, for each visible peak $i = 1, \ldots, n$ let (x_i, y_i) be the on-screen coordinates computed by the application under test, while $(\hat{x_i}, \hat{y_i})$ be the coordinates stored in the gold sequence. The angular error in the position of the i-th peak is defined as:

$$\varepsilon(\hat{x_i}, \hat{y_i}) = \sqrt{d_x(\hat{x_i}, x_i)^2 + d_y(\hat{y_i}, y_i)^2},$$

where

$$d_x(\hat{x}, x) = min(360 - \frac{f}{w}|\hat{x} - x|, \frac{f}{w}|\hat{x} - x|)$$

the angular distance (in degrees) between the tested and gold coordinate along the azimuth axis, given the circular symmetry, f is the horizontal Field Of View (in degrees) of the camera and w is the width (in pixels) of the image. The definition of the angular distance along the roll axis $d_y(\hat{y}, y)$ is similar. The angular error of a whole sequence is defined as the average error over all its frames.

[4]The erroneous omission of a peak may result by, e.g, the wrong computation of peaks occluded by the terrain configuration.

Finally, the **Perceived Quality (PQ)** metric measures the percentage of the frames of a sequence that are "good enough". This indicator can be regarded as the fraction of the entire sequence time during which the user experience was satisfactory. The definition of "good" is based on the other metrics: a frame is good if its average angular error is lower, while peak precision and recall are higher than given thresholds. In the case study, after several experiments, the thresholds have been fixed at 3deg, 0.75 and 0.75 for the three indicators, respectively.

In general, Perceived Quality is the most representative metric at first sight because it summarizes all the other ones. However, low values of the other indicators may be effective in directing the search for a defect.

Frames that do not contain annotated peaks were not considered for the evaluation. Otherwise, metrics such as the Average Angular Error would be computed as 0 for them, which would affect the metric computation rendering it not so realistic.

7.3.4 Experimental results

The testing consists of applying the framework described in Section 7.2 to the gold sequences built as explained in section 7.3.2 to evaluate the application quality. The detection of low values of the indicators signals the insurgence of defects, and the worsening of a value after a software update highlights potential regression errors. Note that regression errors are particularly relevant because the computer vision module at the base of the peak positioning methods contains various complex heuristics and a machine learning submodule, which can be retrained with new data to try and achieve better accuracy, and is configured with multiple parameters, which trade accuracy w.r.t. memory footprint and execution speed. Often a software update aiming at one objective may detriment a conflicting one.

We comment the evaluation of PeakLens for 50 gold sequences and 3 application releases, which were executed on a Google Pixel device. Furthermore, taking into account the fact that a replay is non deterministic and that an intensive usage of a phone may affect its performance, we opted for executing such replays with small pauses programmed in between. The complete evaluation results are reported in Table 7.1. Due to particularly high sensor noise detected at the beginning of the sequences, the first 25 frames of each of them were not considered for the evaluation.

The first application release (SENSOR) represents our baseline, provided that it does not include any intelligent computer vision module analyzing the frames captured by the camera, but just projects the peaks based on the orientation sensor values of the device. The resulting mean and median values of the Average Angular Error and Perceived Quality are rather low, which would probably imply a non satisfactory user experience.

The second application release displayed in the tables reports the indicators for RELEASE A. This version already features the sophisticated Computer Vision module, which significantly improves the performance of the application, achieving an increment in all the overall metrics. Nonetheless, there are a few sequences in which the baseline SENSOR version had a more acceptable performance. By inspecting such sequences we were able to determine that many of these cases

are due to flat terrains and uncertain alignments between frame skyline and terrain. Such cases could be improved by introducing specific heuristics capable of detecting them and of proceeding by just using the orientation sensors.

Finally, the last column of the tables refer to RELEASE B, a version that introduced some modifications regarding the machine learning submodule that detects the mountain skyline for the frames, followed by a different post-processing step that is subsequently aligned w.r.t. the terrain. Overall, the testing framework gave an effective feedback on the new version; the Perceived Quality was not affected w.r.t. the previous version and therefore the performance of the release was considered acceptable.

It is worth to mention that the batch replay evaluations can be immediately aborted to save time in case of detecting the insurgence of obvious defects in the first iterations. In the past, we have experienced such situation when dealing with bugs due to scale factor issues, incorrect vertical offset projections and the manifestation of diverse problems with the computer vision module. PQ decreased strongly in such cases, with sensible angular error increase and loss of both precision and recall. Sequence replay permitted us to locate the wrongly positioned peaks and to remove the defect. The overall results obtained by the testing framework are significantly informative, but so can be the visual inspection of the simulations in order to identify and correct specific bugs that may appear under specific scenarios or conditions.

7.4 Discussion

In this section, we discuss the limits to generalization of the proposed framework by covering the most challenging issues to tackle and we also assess the fidelity of the multi-sensor context simulation by experimenting with a set of mobile devices.

7.4.1 Limits to generalization

The testing framework has been implemented with both the general aim of supporting multi-sensor application testing and with the specific objective of putting it to work in the development and maintenance of a specific application. Retrospectively, the resulting architecture exhibits dependencies on the mobile operating system, on the emulation environment and on the application under test.

The Capture Module of Figure 7.2 is the most general component, depending only on the native sensor APIs of the Operating System. It can be extended to new types of sensors (e.g. temperature) simply by following the steps presented in section 7.2.2. The temporal correlation of the multiple sensor streams is achieved by synchronizing on a master sensor. This policy is normally applied to synchronize on the slowest sensor; however, it is also possible to elect any sensor as the master and synchronize the other streams on its callbacks; for example, one may define the GPS position sensor a the master and read from the other sensors only when an update of the location occurs.

The Replay Module has a dependency on the *virtual* sensor APIs, both in the emulated and on-device execution environment. The Sequence Streamer is coupled to the sensor emulation libraries

	SENSOR					RELEASE A					RELEASE B				
#	Accuracy (%)	Precision (%)	Recall (%)	AAE (°)	PQ (%)	Accuracy (%)	Precision (%)	Recall (%)	AAE (°)	PQ (%)	Accuracy (%)	Precision (%)	Recall (%)	AAE (°)	PQ (%)
1	96,50	100,00	87,16	15,12	0,00	99,62	99,68	99,02	2,06	77,05	96,67	99,26	88,77	6,50	32,00
2	86,09	80,62	73,30	14,60	0,00	97,53	95,56	98,58	2,13	81,18	96,93	94,31	97,42	2,18	81,18
3	89,46	88,14	74,89	15,39	0,00	97,52	98,11	93,95	1,09	99,79	98,87	98,67	97,79	1,42	96,63
4	88,62	84,26	73,00	9,28	10,65	85,79	80,99	72,52	10,07	12,11	80,87	74,09	59,81	8,76	18,64
5	91,68	92,70	87,56	8,68	0,00	97,87	98,45	96,60	1,34	98,52	97,45	97,33	96,66	1,47	95,57
6	96,33	91,95	97,05	3,70	56,84	96,44	90,91	98,02	2,45	78,32	96,67	93,07	96,02	2,60	81,68
7	94,89	87,99	97,82	4,08	27,22	97,11	91,88	99,76	1,20	87,50	97,64	93,92	99,76	1,14	89,44
8	97,80	95,93	96,24	2,81	53,47	99,33	98,34	99,19	1,25	96,21	99,27	98,19	99,20	1,31	97,05
9	94,98	88,93	92,00	7,85	15,58	97,65	92,58	99,32	1,04	100,00	99,53	98,96	99,39	1,18	96,84
10	98,56	97,25	94,86	4,94	0,68	99,22	97,71	98,19	2,09	82,25	99,57	99,01	98,84	1,60	98,63
11	100,00	100,00	100,00	1,92	100,00	93,75	100,00	50,00	0,51	0,00	94,19	100,00	53,51	0,71	7,02
12	46,29	6,39	6,65	59,09	0,00	46,93	10,25	12,34	58,38	0,00	56,69	24,68	25,00	39,94	0,00
13	66,78	31,53	29,71	42,82	0,00	62,04	19,48	19,68	47,33	0,00	64,47	26,44	26,16	45,13	0,00
14	94,54	68,22	96,88	6,09	28,97	93,95	66,24	92,60	6,33	25,70	94,22	67,33	92,99	6,87	25,70
15	70,69	67,30	68,26	20,64	0,00	90,73	90,36	90,35	6,38	67,94	95,87	92,58	99,16	1,72	86,12
16	94,51	86,66	100,00	2,14	93,97	93,46	84,31	100,00	2,04	96,12	93,46	84,74	99,25	1,59	99,14
17	96,62	100,00	91,54	6,62	0,00	94,75	100,00	85,31	9,25	0,00	97,36	100,00	93,40	7,16	7,42
18	98,13	100,00	94,76	3,02	55,86	99,06	98,07	99,31	1,91	96,55	99,26	98,07	99,86	1,68	97,93
19	92,90	81,02	94,83	5,82	0,42	95,92	86,19	99,74	1,34	82,95	95,56	85,45	99,38	1,39	80,63
20	89,32	70,20	94,25	5,69	0,31	95,13	81,73	99,03	1,56	70,22	93,77	79,20	97,60	3,12	52,98
21	97,91	95,30	97,64	3,45	36,42	98,79	97,16	98,96	1,43	98,95	98,72	97,14	98,79	1,58	97,89
22	94,20	92,74	88,33	7,66	0,00	97,49	93,55	98,58	1,73	73,63	97,57	94,27	98,10	1,92	69,62
23	99,52	96,89	100,00	1,60	84,86	99,63	99,59	97,97	4,21	42,70	98,73	98,78	92,97	5,54	39,73
24	96,77	88,73	99,92	2,48	89,95	99,11	96,87	99,92	2,39	80,90	96,63	91,98	94,29	2,08	93,47
25	97,25	81,16	98,37	2,94	46,29	97,21	80,69	100,00	5,41	29,97	97,61	82,49	99,70	5,57	35,31
26	97,27	96,52	88,90	6,14	0,00	97,56	96,10	90,22	8,28	18,39	91,17	90,99	48,32	4,33	5,04
27	94,41	86,50	93,19	3,08	37,37	92,96	89,69	87,15	9,82	18,79	81,07	85,30	47,97	8,32	0,00
28	99,65	99,95	98,92	2,11	100,00	98,27	97,76	95,99	1,25	100,00	99,15	98,55	98,49	1,13	100,00
29	76,34	57,30	56,42	22,09	0,00	77,91	60,38	58,46	20,55	0,00	94,25	83,82	98,53	1,50	98,04
30	93,61	70,42	96,51	5,45	9,42	94,47	69,55	97,03	1,93	23,04	93,61	68,85	94,42	4,45	2,62
31	95,46	92,82	95,76	4,48	29,18	97,64	95,79	98,64	1,51	82,56	96,53	95,20	96,56	1,50	76,16
32	91,37	77,24	94,74	5,32	24,63	90,48	74,82	93,22	5,21	37,68	90,48	75,20	93,73	5,59	34,11
33	93,04	100,00	76,03	3,88	45,88	95,15	100,00	83,53	2,17	76,47	93,87	94,85	86,52	2,97	75,88
34	93,75	66,67	100,00	5,97	0,00	93,75	66,67	100,00	9,35	0,00	99,68	98,31	100,00	11,77	0,00
35	94,07	72,08	100,00	6,40	0,00	94,52	76,06	97,59	0,79	23,10	95,23	77,50	100,00	1,48	28,93
36	92,87	100,00	78,60	23,82	0,00	100,00	100,00	100,00	1,66	99,56	100,00	100,00	100,00	1,14	100,00
37	94,07	82,14	99,83	2,44	66,84	92,69	80,70	94,64	1,52	64,29	93,28	81,63	95,58	1,45	64,80
38	86,54	50,48	100,96	2,20	0,00	86,95	51,44	100,96	6,03	0,00	86,54	50,48	100,96	5,39	0,00
39	96,36	94,36	95,08	4,21	6,37	98,32	97,73	97,46	2,48	85,14	96,83	97,26	93,06	1,63	93,84
40	96,36	89,68	96,09	4,94	2,95	99,43	98,77	99,00	1,33	97,26	99,19	98,21	98,74	1,36	96,63
41	85,50	63,54	88,60	11,39	0,00	90,24	69,33	99,58	1,42	21,47	91,21	73,58	99,72	1,83	26,11
42	85,99	100,00	60,53	17,95	0,00	92,60	99,58	82,07	8,02	50,74	99,61	99,09	99,72	1,54	96,21
43	93,09	89,89	83,47	11,48	0,00	98,32	98,00	96,95	2,66	79,58	99,19	98,95	98,63	1,34	93,89
44	98,35	96,12	96,42	6,03	0,00	99,38	98,02	99,25	1,84	95,58	98,76	94,30	99,49	1,55	88,21
45	92,06	89,40	77,16	19,98	0,00	94,48	89,65	88,53	16,93	0,00	93,05	90,14	80,74	19,83	0,00
46	75,58	52,08	74,39	17,05	0,00	92,91	79,07	100,00	3,70	66,74	93,16	80,41	99,37	2,62	68,63
47	57,05	2,24	2,24	59,49	0,00	59,29	12,18	12,18	42,21	0,00	69,55	31,41	31,41	16,79	23,72
48	76,42	60,14	69,65	18,92	1,68	82,18	71,05	76,81	17,85	17,89	83,75	72,46	79,09	15,80	17,26
49	98,46	100,00	97,06	6,66	0,00	98,91	99,57	98,08	2,94	72,12	98,91	99,79	98,40	1,99	81,09
50	98,37	100,00	98,16	2,34	69,05	94,95	99,86	93,51	2,04	73,68	98,21	99,74	97,96	2,41	69,47
Mean	90,53	81,27	85,07	10,65	21,90	92,79	84,41	88,80	6,97	55,65	93,48	86,12	88,82	5,48	58,43
Median	94,14	88,83	94,75	6,00	0,55	95,54	93,06	97,78	2,15	71,17	96,65	94,09	97,88	1,95	72,75

Table 7.1: *Complete evaluation testing results.*

7.4.2 Limits to fidelity

As in the work of Rege et al. [179], we have evaluated the limits in the reconstruction of realistic working conditions during the testing process. Also, the most significant challenge in our case study is the faithful reproduction of the functioning of the camera sensor, which is the most difficult to simulate. Since the camera sensor is the slowest one, on which the readings of the remaining sensors depend, a prominent goal of the testing process is to achieve a frame replay rate in the testing framework as close as possible to the one observed during the execution of the application in field conditions. This entails that the Capture and Replay modules should be able to acquire frames and replay them at the same rate as the real application.

Table 7.2 reports the camera frame acquisition rates observed in a set of mobile devices, chosen to have a representative range going from medium-end models (LG G5) to high-end (Google Pixel) models. The results may vary depending on the adopted frame size; the size considered in the case study is 640x480 pixels, which is the one normally used in the case of study for the image analysis in mobile devices. In general, the Capture module achieves a slightly higher frame rate than the camera preview in the normal execution of the application. This is due to the fact that the application execution requires more system resources for running the computer vision algorithms. Conversely, the Replay module achieves a lower frame rate. The reason is that camera frames are read from secondary storage, which is slower than the access to the frames from the camera sensor. Despite these differences, the Replay module executed in the mobile device still represents a sufficiently good approximation of the real functioning of the application.

Phone model	App (fps)	Capture (fps)	Replay (fps)
Google Pixel	30	30	27
Motorola Nexus 6	28	29	22
LG Nexus 5x	27	28	20
LG G5 SE	24	30	17

Table 7.2: *Comparison of the frame processing rates in the application, in the Capture module and in the Replay module executed in the mobile device.*

The execution in the Android Emulator with the default configuration parameters yielded an extremely low frame rate for the camera preview (6fps at maximum). The Android Emulator can be configured to exploit hardware acceleration using the Graphics Processing Unit (GPU) of the host workstation, thus achieving a frame rate close to the one observed in field conditions. However, in a data-intensive application as the one presented in the case study, such acceleration alters the computation power of the emulated device, and thus results in a far less realistic simulation. To the best of our knowledge, Android Emulators are still unable to realistically replicate the performance of both camera sensors and processors of real devices, which makes them not yet ideal for testing multi-sensor mobile applications with real-time data processing requirements over camera frame data. For this reason, we did not proceed with the implementation of the Replay module on top of the Emulator environment.

CHAPTER 8

Conclusions and Future Work

In this thesis we explored the feasibility of exploiting the recent progress in the Artificial Intelligence, Computer Vision and Augmented Reality fields in order to provide enhanced outdoor Augmented Reality entertainment/informative solutions. We concluded that such integration can lead to the construction of high quality, successful and engaging applications, suitable for limited portable devices by means of better understanding the surrounding visual context captured by the camera with a marker-less approach executed on-board the devices. We addressed the arising difficulties and validated the proposed solutions through the use case of a real-world outdoor Mobile Augmented Reality application for mountain exploration, which has so far reached +500k installs and can be conveniently used for both improving the awareness about the mountain environment and for crowdsourcing environmental tasks, such as data collection.

We introduced the use of DL models for mountain skyline extraction to deal with accurate image-to-terrain geolocalization and camera orientation estimation in natural mountain environments. Such models were trained with a large set of annotated images taken in uncontrolled conditions, and are capable of identifying obstacles interrupting the skyline with a patch-based approach. Obstacles identification represents an essential element to take into account for a correct alignment w.r.t. the terrain. Therefore, we have defined specific evaluation metrics to measure it. In the future, we will further experiment with efficient alternative approaches to cope with more global-aware context information to complement the local features used in the trained DL models.

We presented a framework for the development of outdoor MAR applications to cope with several difficulties, such as unreliable connection, uncertain positioning, occlusions and real-time requirements; and defined a characterization for the

dimensions of heterogeneous meta-data to augment the view. We also discussed its use for mountain image enrichment based on DEM and GIS data, which had the primary goal of attracting the interest of tourists and enhancing their outdoor experience. In the future, we plan to exploit collected data and incorporate DL techniques to improve the knowledge extraction phase and aggregate such derived information into environmental prediction and decision models.

We tackled the problem of efficiently supporting the optimization and deployment of DL models on-board devices with limited resources and heterogeneous architectures. We have also presented PolimiDL, an open source publicly available framework for the acceleration of DL inference on mobile and embedded systems, which has proved competitive w.r.t. TensorFlow Lite, achieving better performance on a set of small models. Future work will concentrate on adding support for more layers, model quantization, and conversion of ONNX format, to simplify the porting from more DL frameworks. Moreover, experimentation will be extended by evaluating additional models, configurations, metrics (e.g. energy consumption and memory accesses) and devices (e.g. Raspberries and drones).

We presented a capture and replay framework for the automated testing of mobile applications that depend on noisy multiple correlated sensor streams, thus becoming suitable to support the development and maintenance of MAR applications under controlled lab conditions. We have applied and reported the results of using such framework to support the mountain exploration use case, where the input stream is heterogeneous and contains noisy sensor data, and the output is the sequence of 2D coordinates of relevant objects in the camera frames. Future work will concentrate on further generalizing the framework by instantiating it for other use cases, incorporating the capture and replay of GUI events as well, so as to achieve the automated testing of application usage sequences including user's gestures. It will also focus on the integration of the framework with cloud-enabled mobile execution services, such as Amazon AWS Mobile Farm[1], on the experimentation with mobile emulation platforms to achieve a more realistic reproduction of field conditions, and on the construction of a web version of the testing framework whereby developers could execute the entire testing process completely online.

Finally, we plan to continue with this research track, and further optimize DL for its deployment on mobile and embedded systems, in order to provide enhanced experiences and support its incorporation to benefit more real-world applications, motivated by the capabilities and challenges of the Internet of Things (IoT). Experimentation with training and adaptation of models directly on-board device is also of particular interest.

[1] https://aws.amazon.com/device-farm/ [accessed 21 August 2019]

Bibliography

[1] Gregory D. Abowd, Anind K. Dey, Peter J. Brown, Nigel Davies, Mark Smith, and Pete Steggles. Towards a better understanding of context and context-awareness. In Hans-Werner Gellersen, editor, *Handheld and Ubiquitous Computing, First International Symposium, HUC'99, Karlsruhe, Germany, September 27-29, 1999, Proceedings*, volume 1707 of *Lecture Notes in Computer Science*, pages 304–307. Springer, 1999.

[2] Touqeer Ahmad, George Bebis, Monica Nicolescu, Ara Nefian, and Terry Fong. An edge-less approach to horizon line detection. In *Machine Learning and Applications (ICMLA), 2015 IEEE 14th International Conference on*, pages 1095–1102. IEEE, 2015.

[3] Touqeer Ahmad, Pavel Campr, Martin Čadik, and George Bebis. Comparison of semantic segmentation approaches for horizon/sky line detection. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 4436–4443. IEEE, 2017.

[4] Shaikhah AlEbrahim and Imtiaz Ahmad. Task scheduling for heterogeneous computing systems. *The Journal of Supercomputing*, 73(6):2313–2338, 2017.

[5] Mohammad Allahbakhsh, Boualem Benatallah, Aleksandar Ignjatovic, Hamid Reza Motahari-Nezhad, Elisa Bertino, and Schahram Dustdar. Quality control in crowdsourcing systems: Issues and directions. *IEEE Internet Computing*, 17(2):76–81, 2013.

[6] Dhiraj Amin and Sharvari Govilkar. Comparative study of augmented reality sdks. *International Journal on Computational Science & Applications*, 5(1):11–26, 2015.

[7] Andrew Anderson, Aravind Vasudevan, Cormac Keane, and David Gregg. Low-memory gemm-based convolution algorithms for deep neural networks. *arXiv preprint arXiv:1709.03395*, 2017.

[8] Anil Armagan, Martin Hirzer, Peter M Roth, and Vincent Lepetit. Learning to align semantic segmentation and 2.5 d maps for geolocalization. In *Conference on Computer Vision and Pattern Recognition*, 2017.

[9] Nils Aschenbruck, Raphael Ernst, Elmar Gerhards-Padilla, and Matthias Schwamborn. Bonnmotion: A mobility scenario generation and analysis tool. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 51:1–51:10, ICST, Brussels, Belgium, Belgium, 2010. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[10] Ronald T Azuma. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, 6(4):355–385, 1997.

[11] Jimmy Ba and Rich Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.

[12] Georges Baatz, Olivier Saurer, Kevin Köser, and Marc Pollefeys. Large scale visual geo-localization of images in mountainous terrain. In *Computer Vision–ECCV 2012*. 2012.

[13] Lionel Baboud, Martin Čadík, Elmar Eisemann, and Hans-Peter Seidel. Automatic photo-to-terrain alignment for the annotation of mountain pictures. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on*, pages 41–48. IEEE, 2011.

[14] Jorge Bacca, Silvia Baldiris, Ramon Fabregat, Sabine Graf, et al. Augmented reality trends in education: a systematic review of research and applications. *Journal of Educational Technology & Society*, 17(4):133, 2014.

[15] Bowen Baker, Otkrist Gupta, Nikhil Naik, and Ramesh Raskar. Designing neural network architectures using reinforcement learning. *arXiv preprint arXiv:1611.02167*, 2016.

[16] Mayank Bansal and Kostas Daniilidis. Geometric urban geo-localization. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2014.

[17] Youcef Bentoutou, Nasreddine Taleb, Kidiyo Kpalma, and Joseph Ronsin. An automatic image registration for applications in remote sensing. *IEEE Trans. Geoscience and Remote Sensing*, 43(9):2127–2137, 2005.

[18] Claudio Bettini, Oliver Brdiczka, Karen Henricksen, Jadwiga Indulska, Daniela Nicklas, Anand Ranganathan, and Daniele Riboni. A survey of context modelling and reasoning techniques. *Pervasive and Mobile Computing*, 6(2):161–180, 2010.

[19] Sourav Bhattacharya and Nicholas D Lane. Sparsification and separation of deep learning layers for constrained resource inference on wearables. In *Proceedings of the 14th ACM Conference on Embedded Network Sensor Systems, SenSys 2016, Stanford, CA, USA, November 14-16, 2016*, pages 176–189, 2016.

[20] Mark Bilandzic, Michael Banholzer, Deyan Peev, Vesko Georgiev, Florence Balagtas-Fernandez, and Alexander De Luca. Laermometer: a mobile noise mapping application. In *Proceedings of the 5th Nordic Conference on Human-computer interaction: building bridges*, pages 415–418. ACM, 2008.

[21] Mark Billinghurst, Adrian J. Clark, and Gun A. Lee. A survey of augmented reality. *Foundations and Trends in Human-Computer Interaction*, 8(2-3):73–272, 2015.

[22] Christopher M Bishop. *Pattern recognition and machine learning*. springer, 2006.

[23] Christopher M Bishop et al. *Neural networks for pattern recognition*. Oxford university press, 1995.

[24] Jeffrey R Blum, Daniel G Greencorn, and Jeremy R Cooperstock. Smartphone sensor reliability for augmented reality applications. In *International Conference on Mobile and Ubiquitous Systems: Computing, Networking, and Services*, pages 127–138. Springer, 2012.

[25] Jan Brejcha and Martin Čadík. Geopose3k: Mountain landscape dataset for camera pose estimation in outdoor environments. *Image and Vision Computing*, 66:1–14, 2017.

[26] Jan Brejcha and Martin Čadík. State-of-the-art in visual geo-localization. *Pattern Analysis and Applications*, 20(3):613–637, 2017.

[27] Kai Briechle and Uwe D Hanebeck. Template matching using fast normalized cross correlation. In *Aerospace/Defense Sensing, Simulation, and Controls*, pages 95–102. International Society for Optics and Photonics, 2001.

[28] Cristian Bucilua, Rich Caruana, and Alexandru Niculescu-Mizil. Model compression. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 535–541. ACM, 2006.

[29] Michael Buhrmester, Tracy Kwang, and Samuel D Gosling. Amazon's mechanical turk: A new source of inexpensive, yet high-quality, data? *Perspectives on psychological science*, 6(1):3–5, 2011.

[30] Ivan Cabrilo, Philippe Bijlenga, and Karl Schaller. Augmented reality in the surgery of cerebral arteriovenous malformations: technique assessment and considerations. *Acta neurochirurgica*, 156(9):1769–1774, 2014.

[31] Roberta Calegari, Mirco Musolesi, Franco Raimondi, and Cecilia Mascolo. Ctg: A connectivity trace generator for testing the performance of opportunistic mobile systems. In *Proceedings of the the 6th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, pages 415–424. ACM, 2007.

[32] Andrea Castelletti, Roman Fedorov, Piero Fraternali, and Matteo Giuliani. Multimedia on the mountaintop: Using public snow images to improve water systems operation. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 948–957. ACM, 2016.

[33] Claudio Cavallaro, Roman Fedorov, Carlo Bernaschina, and Piero Fraternali. Compressing web geodata for real-time environmental applications. In *International Workshop on the Internet for Financial Collective Awareness and Intelligence*, pages 119–128. Springer International Publishing, 2016.

[34] Boris Y Chan, Antonio Si, and Hong Va Leong. Cache management for mobile databases: Design and evaluation. In *Data Engineering, 1998. Proceedings., 14th International Conference on*, pages 54–63. IEEE, 1998.

[35] Chenyi Chen, Ari Seff, Alain Kornhauser, and Jianxiong Xiao. Deepdriving: Learning affordance for direct perception in autonomous driving. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2722–2730, 2015.

[36] Guanling Chen and David Kotz. A survey of context-aware mobile computing research. Technical report, Hanover, NH, USA, 2000.

[37] Liang-Chieh Chen, George Papandreou, Iasonas Kokkinos, Kevin Murphy, and Alan L Yuille. Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs. *IEEE transactions on pattern analysis and machine intelligence*, 40(4):834–848, 2018.

[38] Tianshi Chen, Zidong Du, Ninghui Sun, Jia Wang, Chengyong Wu, Yunji Chen, and Olivier Temam. Diannao: A small-footprint high-throughput accelerator for ubiquitous machine-learning. In *ACM Sigplan Notices*, volume 49, pages 269–284. ACM, 2014.

[39] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[40] Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang. A survey of model compression and acceleration for deep neural networks. *arXiv preprint arXiv:1710.09282*, 2017.

[41] Hyun-Duk Cho, Ph D Principal Engineer, Kisuk Chung, and Taehoon Kim. Benefits of the big. little architecture. *EETimes, Feb*, 2012.

[42] François Chollet. Xception: Deep learning with depthwise separable convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1251–1258, 2017.

[43] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, and Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. In *Int. Conf. on Medical Image Computing and Computer-assisted Intervention*, pages 411–418. Springer, 2013.

[44] Marius Cordts, Mohamed Omran, Sebastian Ramos, Timo Rehfeld, Markus Enzweiler, Rodrigo Benenson, Uwe Franke, Stefan Roth, and Bernt Schiele. The cityscapes dataset for semantic urban scene understanding. In *Proc. of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[45] Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

[46] David J Crandall, Lars Backstrom, Daniel Huttenlocher, and Jon Kleinberg. Mapping the world's photos. In *Proceedings of the 18th international conference on World wide web*, pages 761–770. ACM, 2009.

[47] Scott G Dacko. Enabling smart retail settings via mobile augmented reality shopping apps. *Technological Forecasting and Social Change*, 124:243–256, 2017.

[48] Patrick Dähne and John N Karigiannis. Archeoguide: System architecture of a mobile outdoor augmented reality system. In *null*, page 263. IEEE, 2002.

[49] Stefan Daume and Victor Galaz. "anyone know what species this is?"–twitter conversations as embryonic citizen science communities. *PloS one*, 11(3):e0151387, 2016.

[50] Ranieri de Brito Moreira, Lıvia Castro Degrossi, and Joao Porto de Albuquerque. An experimental evaluation of a crowdsourcing-based approach for flood risk management. In *Paper presented at the Conference: 12th Workshop on Experimental Software Engineering (ESELAW), at Lima, Peru*, 2015.

[51] Lívia Castro Degrossi, JP Albuquerque, Maria Clara Fava, and Eduardo Mario Mendiondo. Flood citizen observatory: a crowdsourcing-based approach for flood risk management in brazil. In *26th Int. Conf. on Software Engineering and Knowledge Engineering*, 2014.

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

[53] Emily L Denton, Wojciech Zaremba, Joan Bruna, Yann LeCun, and Rob Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in neural information processing systems*, pages 1269–1277, 2014.

[54] Andreas Juergen Dietz, Claudia Kuenzer, Ursula Gessner, and Stefan Dech. Remote sensing of snow–a review of available methods. *International Journal of Remote Sensing*, 33(13):4094–4134, 2012.

[55] Hoang T Dinh, Chonho Lee, Dusit Niyato, and Ping Wang. A survey of mobile cloud computing: architecture, applications, and approaches. *Wireless communications and mobile computing*, 13(18):1587–1611, 2013.

[56] Céline Dizerens, Fabia Hüsler, and Stefan Wunderle. Webcam imagery rectification and classification: Potential for complementing satellite-derived snow maps over switzerland.

[57] Omar El Ariss, Dianxiang Xu, Santosh Dandey, Brad Vender, Phil McClean, and Brian Slator. A systematic capture and replay strategy for testing complex gui based java applications. In *Information Technology: New Generations (ITNG), 2010 Seventh International Conference on*, pages 1038–1043. IEEE, 2010.

[58] Enrique Estellés-Arolas and Fernando González-Ladrón-De-Guevara. Towards an integrated crowdsourcing definition. *Journal of Information science*, 38(2):189–200, 2012.

[59] Andre Esteva, Brett Kuprel, Roberto A Novoa, Justin Ko, Susan M Swetter, Helen M Blau, and Sebastian Thrun. Dermatologist-level classification of skin cancer with deep neural networks. *Nature*, 542(7639):115–118, 2017.

[60] Roman Fedorov, Darian Frajberg, and Piero Fraternali. A framework for outdoor mobile augmented reality and its application to mountain peak detection. In *International Conference on Augmented Reality, Virtual Reality and Computer Graphics*, pages 281–301. Springer, 2016.

[61] Roman Fedorov, Piero Fraternali, and Chiara Pasini. Snowwatch: a multi-modal citizen science application. In *Web Engineering*. 2016.

[62] Colin J Ferster and Nicholas C Coops. A review of earth observation using mobile personal communication devices. *Computers & Geosciences*, 51:339–349, 2013.

[63] Alexander Finkelstein, Uri Almog, and Mark Grobman. Fighting quantization bias with bias. *arXiv preprint arXiv:1906.03193*, 2019.

[64] Darian Frajberg, Carlo Bernaschina, Christian Marone, and Piero Fraternali. Accelerating deep learning inference on mobile systems. *To be published in International Conference on AI & Mobile Services*, 2019.

[65] Darian Frajberg, Piero Fraternali, and Rocio Nahime Torres. Convolutional neural network for pixel-wise skyline detection. In *International Conference on Artificial Neural Networks*, pages 12–20. Springer, 2017.

[66] Darian Frajberg, Piero Fraternali, and Rocio Nahime Torres. Heterogeneous information integration for mountain augmented reality mobile apps. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 313–322. IEEE, 2017.

[67] Darian Frajberg, Piero Fraternali, Rocio Nahime Torres, Carlo Bernaschina, and Roman Fedorov. A testing framework for multi-sensor mobile applications. *To be published in Journal of Mobile Multimedia*, 2019.

[68] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[69] Hervé Goëau, Pierre Bonnet, Alexis Joly, Vera Bakić, Julien Barbe, Itheri Yahiaoui, Souheil Selmi, Jennifer Carré, Daniel Barthélémy, Nozha Boujemaa, et al. Pl@ntnet mobile app. In *Proceedings of the 21st International Conference on Multimedia*. ACM, 2013.

[70] Lorenzo Gomez, Iulian Neamtiu, Tanzirul Azim, and Todd Millstein. Reran: Timing-and touch-sensitive record and replay for android. In *2013 35th International Conference on Software Engineering (ICSE)*, pages 72–81. IEEE, 2013.

[71] Yunchao Gong, Liu Liu, Ming Yang, and Lubomir Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

[72] Michael F Goodchild. Citizens as sensors: the world of volunteered geography. *GeoJournal*, 69(4):211–221, 2007.

[73] Yanming Guo, Yu Liu, Ard Oerlemans, Songyang Lao, Song Wu, and Michael S Lew. Deep learning for visual understanding: A review. *Neurocomputing*, 187:27–48, 2016.

[74] Suyog Gupta, Ankur Agrawal, Kailash Gopalakrishnan, and Pritish Narayanan. Deep learning with limited numerical precision. In *International Conference on Machine Learning*, pages 1737–1746, 2015.

[75] Philipp Gysel. Ristretto: Hardware-oriented approximation of convolutional neural networks. *arXiv preprint arXiv:1605.06402*, 2016.

[76] Mordechai Haklay and Patrick Weber. Openstreetmap: User-generated street maps. *IEEE Pervasive Computing*, 7(4):12–18, 2008.

[77] Song Han, Junlong Kang, Huizi Mao, Yiming Hu, Xin Li, Yubin Li, Dongliang Xie, Hong Luo, Song Yao, Yu Wang, et al. Ese: Efficient speech recognition engine with sparse lstm on fpga. In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pages 75–84. ACM, 2017.

[78] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A Horowitz, and William J Dally. Eie: efficient inference engine on compressed deep neural network. In *2016 ACM/IEEE 43rd Annual International Symposium on Computer Architecture (ISCA)*, pages 243–254. IEEE, 2016.

[79] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015.

[80] Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[81] Jussi Hanhirova, Teemu Kämäräinen, Sipi Seppälä, Matti Siekkinen, Vesa Hirvisalo, and Antti Ylä-Jääski. Latency and throughput characterization of convolutional neural networks for mobile computer vision. In *Proceedings of the 9th ACM Multimedia Systems Conference*, pages 204–215. ACM, 2018.

[82] Babak Hassibi, David G Stork, and Gregory J Wolff. Optimal brain surgeon and general network pruning. In *IEEE international conference on neural networks*, pages 293–299. IEEE, 1993.

[83] Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

[84] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 2961–2969, 2017.

[85] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[86] 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.

[87] Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, and Song Han. Amc: Automl for model compression and acceleration on mobile devices. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 784–800, 2018.

[88] Yihui He, Xiangyu Zhang, and Jian Sun. Channel pruning for accelerating very deep neural networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1389–1397, 2017.

[89] Matthias Hentschel and Bernardo Wagner. Autonomous robot navigation based on openstreetmap geodata. In *13th International IEEE Conference on Intelligent Transportation Systems*, pages 1645–1650. IEEE, 2010.

[90] Derek L G Hill, Philipp G Batchelor, Mark Holden, and David J Hawkes. Medical image registration. *Physics in Medicine and Biology*, 46(3):R1–R45, feb 2001.

[91] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal processing magazine*, 29, 2012.

[92] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[93] Catherine Hoffman, Caren B Cooper, Eric B Kennedy, Mahmud Farooque, and Darlene Cavalier. Scistarter 2.0: A digital platform to foster and study sustained engagement in citizen science. In *Analyzing the Role of Citizen Science in Modern Research*, pages 50–61. IGI Global, 2017.

[94] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017.

[95] Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[96] Jung-Chang Huang and Tau Leng. Generalized loop-unrolling: a method for program speedup. In *Symposium on Application-Specific Systems and Software Engineering and Technology. ASSET'99 (Cat. No. PR00122)*, pages 244–248. IEEE, 1999.

[97] Xinyu Huang, Xinjing Cheng, Qichuan Geng, Binbin Cao, Dingfu Zhou, Peng Wang, Yuanqing Lin, and Ruigang Yang. The apolloscape dataset for autonomous driving. *arXiv preprint arXiv:1803.06184*, 2018.

[98] Yanping Huang, Yonglong Cheng, Dehao Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. *arXiv preprint arXiv:1811.06965*, 2018.

[99] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, 2017.

[100] Yao-Ling Hung, Chih-Wen Su, Yuan-Hsiang Chang, Jyh-Chian Chang, and Hsiao-Rong Tyan. Skyline localization for mountain images. In *Multimedia and Expo (ICME), 2013 IEEE International Conference on*, pages 1–6. IEEE, 2013.

[101] Loc N Huynh, Youngki Lee, and Rajesh Krishna Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *15th Annual International Conference on Mobile Systems, Applications, and Services*, pages 82–95. ACM, 2017.

[102] Jyh-Jing Hwang and Tyng-Luh Liu. Pixel-wise deep learning for contour detection. *arXiv preprint arXiv:1504.01989*, 2015.

[103] Kyuyeon Hwang and Wonyong Sung. Fixed-point feedforward deep neural network design using weights+ 1, 0, and- 1. In *Signal Processing Systems (SiPS), 2014 IEEE Workshop on*, pages 1–6. IEEE, 2014.

[104] Otto Hyvärinen and Elena Saltikoff. Social media as a source of meteorological observations. *Monthly Weather Review*, 138(8):3175–3184, 2010.

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

[106] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *European Conference on Computer Vision*, pages 288–314. Springer, 2018.

[107] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[108] José Luis Izkara, Juan Pérez, Xabier Basogain, and Diego Borro. Mobile augmented reality, an advanced tool for the construction sector. In *Proceedings of the 24th W78 conference, Maribor, Slovenia*, pages 190–202. Citeseer, 2007.

[109] Max Jaderberg, Andrea Vedaldi, and Andrew Zisserman. Speeding up convolutional neural networks with low rank expansions. *arXiv preprint arXiv:1405.3866*, 2014.

[110] Puneet Jain, Justin Manweiler, and Romit Roy Choudhury. Overlay: Practical mobile augmented reality. In *Proceedings of the 13th Annual International Conference on Mobile Systems, Applications, and Services*, pages 331–344. ACM, 2015.

[111] Ana Javornik. Augmented reality: Research agenda for studying the impact of its media characteristics on consumer behaviour. *Journal of Retailing and Consumer Services*, 30:252–261, 2016.

[112] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[113] Alexis Joly, Hervé Goëau, Julien Champ, Samuel Dufour-Kowalski, Henning Müller, and Pierre Bonnet. Crowdsourcing biodiversity monitoring: how sharing your photo stream can sustain our planet. In *Proceedings of the 24th ACM international conference on Multimedia*, pages 958–967. ACM, 2016.

[114] Shrinivas Joshi and Alessandro Orso. SCARPE: A technique and tool for selective capture and replay of program executions. In *23rd IEEE International Conference on Software Maintenance (ICSM 2007), October 2-5, 2007, Paris, France*, pages 234–243. IEEE, 2007.

[115] Stephan Karpischek, Claudio Marforio, Mike Godenzi, Stephan Heuel, and Florian Michahelles. Swisspeaks–mobile augmented reality to identify mountains. In *Workshop at the International Symposium on Mixed and Augmented Reality 2009 (ISMAR 2009)*. Citeseer, 2009.

[116] Felix Keis and Kevin Wiesner. Participatory sensing utilized by an advanced meteorological nowcasting system. In *Intelligent Sensors, Sensor Networks and Information Processing (ISSNIP), 2014 IEEE Ninth International Conference on*, pages 1–6. IEEE, 2014.

[117] Sunyoung Kim, Jennifer Mankoff, and Eric Paulos. Sensr: evaluating a flexible framework for authoring mobile data-collection tools for citizen science. In *Proceedings of the 2013 conference on Computer supported cooperative work*, pages 1453–1462. ACM, 2013.

[118] Yong-Deok Kim, Eunhyeok Park, Sungjoo Yoo, Taelim Choi, Lu Yang, and Dongjun Shin. Compression of deep convolutional neural networks for fast and low power mobile applications. *arXiv preprint arXiv:1511.06530*, 2015.

[119] Qingkai Kong, Richard M Allen, Louis Schreier, and Young-Woo Kwon. Myshake: A smartphone seismic network for earthquake early warning and beyond. *Science advances*, 2(2):e1501055, 2016.

[120] Panos Kourouthanassis, Costas Boletsis, Cleopatra Bardaki, and Dimitra Chasanidou. Tourists responses to mobile augmented reality travel guides: The role of emotions on adoption behavior. *Pervasive and Mobile Computing*, 18:71–87, 2015.

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

[122] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, Lei Jiao, Lorena Qendro, and Fahim Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proceedings of the 15th International Conference on Information Processing in Sensor Networks*, page 23. IEEE Press, 2016.

[123] Nicholas D Lane, Sourav Bhattacharya, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. An early resource characterization of deep learning on wearables, smartphones and internet-of-things devices. In *Proceedings of the 2015 international workshop on internet of things towards applications*, pages 7–12. ACM, 2015.

[124] Nicholas D Lane, Sourav Bhattacharya, Akhil Mathur, Petko Georgiev, Claudio Forlivesi, and Fahim Kawsar. Squeezing deep learning into mobile and embedded devices. *IEEE Pervasive Computing*, 16(3):82–88, 2017.

[125] Nicholas D Lane and Petko Georgiev. Can deep learning revolutionize mobile sensing? In *Proceedings of the 16th International Workshop on Mobile Computing Systems and Applications*, pages 117–122. ACM, 2015.

[126] Nicholas D Lane, Petko Georgiev, and Lorena Qendro. Deepear: robust smartphone audio sensing in unconstrained acoustic environments using deep learning. In *Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 283–294. ACM, 2015.

[127] Nicholas D Lane and Pete Warden. The deep (learning) transformation of mobile and embedded computing. *Computer*, 51(5):12–16, 2018.

[128] G LeBuhn and R Schmucki. Identifying pollination service hotspots and coldspots using citizen science data from the great sunflower project. In *AGU Fall Meeting Abstracts*, 2016.

[129] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436, 2015.

[130] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989.

[131] Yann LeCun, John S Denker, and Sara A Solla. Optimal brain damage. In *Advances in neural information processing systems*, pages 598–605, 1990.

[132] Dawei Li, Xiaolong Wang, and Deguang Kong. Deeprebirth: Accelerating deep neural network execution on mobile devices. In *Thirty-Second AAAI Conference on Artificial Intelligence*, 2018.

[133] Hao Li, Asim Kadav, Igor Durdanovic, Hanan Samet, and Hans Peter Graf. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*, 2016.

[134] Chieh-Jan Mike Liang, Nicholas D. Lane, Niels Brouwers, Li Zhang, Börje Karlsson, Hao Liu, Yan Liu, Jun Tang, Xiang Shan, Ranveer Chandra, and Feng Zhao. Caiipa: automated large-scale mobile app testing through contextual fuzzing. In Sung-Ju Lee, Ashutosh Sabharwal, and Prasun Sinha, editors, *The 20th Annual International Conference on Mobile Computing and Networking, MobiCom'14, Maui, HI, USA, September 7-11, 2014*, pages 519–530. ACM, 2014.

[135] Wen-Nung Lie, Tom C.-I. Lin, Ting-Chih Lin, and Keng-Shen Hung. A robust dynamic programming algorithm to extract skyline in images for navigation. *Pattern Recognition Letters*, 26(2):221 – 230, 2005.

[136] Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[137] María Teresa Linaza, Aitor Gutierrez, and Ander García. Pervasive augmented reality games to experience tourism destinations. In *Information and Communication Technologies in Tourism 2014*, pages 497–509. Springer, 2013.

[138] Haibin Ling. Augmented reality in reality. *IEEE MultiMedia*, 24(3):10–15, 2017.

[139] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, Francesco Ciompi, Mohsen Ghafoorian, Jeroen Awm Van Der Laak, Bram Van Ginneken, and Clara I Sánchez. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, 2017.

[140] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

[141] Scott R Loss, Sara S Loss, Tom Will, and Peter P Marra. Linking place-based citizen science with large-scale conservation research: a case study of bird-building collisions and the role of professional scientists. *Biological Conservation*, 184:439–445, 2015.

[142] Christopher S Lowry and Michael N Fienen. Crowdhydrology: crowdsourcing hydrologic data and engaging citizen scientists. *Ground Water*, 51(1):151–156, 2013.

[143] Zongqing Lu, Swati Rallapalli, Kevin Chan, and Thomas La Porta. Modeling the resource requirements of convolutional neural networks on mobile devices. *arXiv preprint arXiv:1709.09503*, 2017.

[144] Ningning Ma, Xiangyu Zhang, Hai-Tao Zheng, and Jian Sun. Shufflenet v2: Practical guidelines for efficient cnn architecture design. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 116–131, 2018.

[145] Aravind Machiry, Rohan Tahiliani, and Mayur Naik. Dynodroid: An input generation system for android apps. In *Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering*, ESEC/FSE 2013, pages 224–234, New York, NY, USA, 2013. ACM.

[146] Nicolas Maisonneuve, Matthias Stevens, Maria E Niessen, and Luc Steels. Noisetube: Measuring and mapping noise pollution with mobile phones. In *Information technologies in environmental engineering*, pages 215–228. Springer, 2009.

[147] Sotiris Makris, Panagiotis Karagiannis, Spyridon Koukas, and Aleksandros-Stereos Matthaiakis. Augmented reality system for operator support in human–robot collaborative assembly. *CIRP Annals*, 65(1):61–64, 2016.

[148] Irene Garcia Martí, Luis E Rodríguez, Mauricia Benedito, Sergi Trilles, Arturo Beltrán, Laura Díaz, and Joaquín Huerta. Mobile application for noise pollution monitoring through gamification techniques. In *International Conference on Entertainment Computing*, pages 562–571. Springer, 2012.

[149] Nargess Memarsadeghi. Citizen science [guest editors' introduction]. *Computing in Science Engineering*, 17(4):8–10, July 2015.

[150] Julien Minet, Yannick Curnel, Anne Gobin, Jean-Pierre Goffart, Francois Melard, Bernard Tychon, Joost Wellens, and Pierre Defourny. Crowdsourcing for agricultural applications: A review of uses and opportunities for a farmsourcing approach. *Computers and Electronics in Agriculture*, 142:126–138, 2017.

[151] Anastasia Moumtzidou, Symeon Papadopoulos, Stefanos Vrochidis, Ioannis Kompatsiaris, Konstantinos Kourtidis, George Hloupis, Ilias Stavrakas, Konstantina Papachristopoulou, and Christodoulos Keratidis. Towards air quality estimation using collected multimodal environmental data. In *International Workshop on Internet and Social Media for Environmental Monitoring*. Springer, 2016.

[152] Gustavo Maglhaes Moura and Rodrigo Luis De Souza Da Silva. Analysis and evaluation of feature detection and tracking techniques using open cv with focus on markerless augmented reality applications. *J. Mobile Multimedia*, 12(3&4):291–302, 2017.

[153] Henry Muccini, Antonio Di Francesco, and Patrizio Esposito. Software testing of mobile applications: Challenges and future research directions. In *Proceedings of the 7th International Workshop on Automation of Software Test*, pages 29–35. IEEE Press, 2012.

[154] Haripriya Mukundarajan, Felix Jan Hein Hol, Erica Araceli Castillo, Cooper Newby, and Manu Prakash. Using mobile phones as acoustic sensors for high-throughput mosquito surveillance. *Elife*, 6:e27854, 2017.

[155] Calvin Murdock, Nathan Jacobs, and Robert Pless. Webcam2satellite: Estimating cloud maps from webcam imagery. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 214–221. IEEE, 2013.

[156] Vinod Nair and Geoffrey E Hinton. Rectified linear units improve restricted boltzmann machines. In *Proceedings of the 27th international conference on machine learning (ICML-10)*, pages 807–814, 2010.

[157] February NatureServe. Natureserve explorer: an online encyclopedia of life, 2012.

[158] Greg Newman, Jim Graham, Alycia Crall, and Melinda Laituri. The art and science of multi-scale citizen science support. *Ecological Informatics*, 6(3-4):217–227, 2011.

[159] Hung Nguyen, Sarah J Maclagan, Tu Dinh Nguyen, Thin Nguyen, Paul Flemons, Kylie Andrews, Euan G Ritchie, and Dinh Phung. Animal recognition and identification with deep convolutional neural networks for automated wildlife monitoring. In *2017 IEEE International Conference on Data Science and Advanced Analytics (DSAA)*, pages 40–49. IEEE, 2017.

[160] Heidi Liljeblad Ødegård, Jo Eidsvik, and Stein-Erik Fleten. Value of information analysis of snow measurements for the scheduling of hydropower production. *Energy Systems*, 10(1):1–19, 2019.

[161] Janne Paavilainen, Hannu Korhonen, Kati Alha, Jaakko Stenros, Elina Koskinen, and Frans Mayra. The pokémon go experience: A location-based augmented reality mobile game goes mainstream. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2493–2498. ACM, 2017.

[162] Yangil Park and Jengchung V Chen. Acceptance and adoption of the innovative use of smartphone. *Industrial Management & Data Systems*, 107(9):1349–1365, 2007.

[163] Dexmont Pena, Andrew Forembski, Xiaofan Xu, and David Moloney. Benchmarking of cnns for low-cost, low-power robotics applications. In *RSS 2017 Workshop: New Frontier for Deep Learning in Robotics*, 2017.

[164] Nathan Piasco, Désiré Sidibé, Cédric Demonceaux, and Valérie Gouet-Brunet. A survey on visual-based localization: On the benefit of heterogeneous data. *Pattern Recognition*, 74:90–109, 2018.

[165] Gian Pietro Picco, Christine Julien, Amy L. Murphy, Mirco Musolesi, and Gruia-Catalin Roman. Software engineering for mobility: reflecting on the past, peering into the future. In James D. Herbsleb and Matthew B. Dwyer, editors, *Proceedings of the on Future of Software Engineering, FOSE 2014, Hyderabad, India, May 31 - June 7, 2014*, pages 13–28. ACM, 2014.

[166] Lara Piccolo, Miriam Fernández, Harith Alani, Arno Scharl, Michael Föls, and David Herring. Climate change engagement: Results of a multi-task game with a purpose. In *Proceedings of the 1st international workshop on Social Web for Environmental and Ecological Monitoring*. AAAI Publications, 2016.

[167] Matthew Pittman and Kim Sheehan. Amazonâs mechanical turk a digital sweatshop? transparency and accountability in crowdsourced online research. *Journal of media ethics*, 31(4):260–262, 2016.

[168] Lorenzo Porzi, Samuel Rota Buló, Paolo Valigi, Oswald Lanz, and Elisa Ricci. Learning contours for automatic annotations of mountains pictures on a smartphone. In *Proceedings of the International Conference on Distributed Smart Cameras*, page 13. ACM, 2014.

[169] Lorenzo Porzi, Samuel Rota Bulò, and Elisa Ricci. A deeply-supervised deconvolutional network for horizon line detection. In *Proc. ACM Multimedia Conf.*, pages 137–141. ACM, 2016.

[170] Kathrin Poser and Doris Dransch. Volunteered geographic information for disaster management with application to rapid flood damage estimation. *Geomatica*, 2010.

[171] Rudra PK Poudel, Stephan Liwicki, and Roberto Cipolla. Fast-scnn: fast semantic segmentation network. *arXiv preprint arXiv:1902.04502*, 2019.

[172] Khandaker Mustakimur Rahman, Tauhidul Alam, and Mahfuzulhoq Chowdhury. Location based early disaster warning and evacuation system on mobile phones using openstreetmap. In *2012 IEEE Conference on Open Systems*, pages 1–6. IEEE, 2012.

[173] Pranav Rajpurkar, Jeremy Irvin, Kaylie Zhu, Brandon Yang, Hershel Mehta, Tony Duan, Daisy Ding, Aarti Bagul, Curtis Langlotz, Katie Shpanskaya, et al. Chexnet: Radiologist-level pneumonia detection on chest x-rays with deep learning. *arXiv preprint arXiv:1711.05225*, 2017.

[174] Deva Ramanan and Xiangxin Zhu. Face detection, pose estimation, and landmark localization in the wild. In *Proceedings of the 2012 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2879–2886. Citeseer, 2012.

[175] Mohammad Rastegari, Vicente Ordonez, Joseph Redmon, and Ali Farhadi. Xnor-net: Imagenet classification using binary convolutional neural networks. In *European Conference on Computer Vision*, pages 525–542. Springer, 2016.

[176] Philipp A Rauschnabel, Alexander Rossmann, and M Claudia tom Dieck. An adoption framework for mobile augmented reality games: The case of pokémon go. *Computers in Human Behavior*, 2017.

[177] Lenin Ravindranath, Suman Nath, Jitendra Padhye, and Hari Balakrishnan. Automatic and scalable fault detection for mobile applications. In *Proceedings of the 12th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '14, pages 190–203, New York, NY, USA, 2014. ACM.

[178] Sasank Reddy, Katie Shilton, Jeff Burke, Deborah Estrin, Mark Hansen, and Mani Srivastava. Evaluating participation and performance in participatory sensing. *UrbanSense08*, page 1, 2008.

[179] Manoj R Rege, Vlado Handziski, and Adam Wolisz. Realistic context generation for mobile app testing and performance evaluation. In *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*, pages 297–308. IEEE, 2017.

[180] Gerhard Reitmayr and Tom Drummond. Going out: robust model-based tracking for outdoor augmented reality. In *Proceedings of the 5th IEEE and ACM International Symposium on Mixed and Augmented Reality*, 2006.

[181] Jun Rekimoto and Katashi Nagao. The world through the computer: Computer augmented interaction with real world environments. In *Proceedings of User Interface Software and Technologyâ95*, 1999.

[182] LiKamWa Robert, Hou Yunhui, Gao Yuan, Polansky Mia, and Zhong Lin. Redeye: Analog convnet image sensor architecture for continuous mobile vision. In *43rd ACM/IEEE Annual International Symposium on Computer Architecture, ISCA 2016, Seoul, South Korea, June 18-22, 2016*, pages 255–266, 2016.

[183] Lior Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.

[184] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical Image Computing and Computer-Assisted Intervention*, pages 234–241. Springer, 2015.

[185] Dominic Rüfenacht, Matthew Brown, Jan Beutel, and Sabine Süsstrunk. Temporally consistent snow cover estimation from noisy, irregularly sampled measurements. In *Proc. 9th International Conference on Computer Vision Theory and Applications*, 2014.

[186] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[187] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. Imagenet large scale visual recognition challenge. *International journal of computer vision*, 115(3):211–252, 2015.

[188] Stuart J Russell and Peter Norvig. *Artificial intelligence: a modern approach*. Malaysia; Pearson Education Limited,, 2016.

[189] Takeshi Sakaki, Makoto Okazaki, and Yutaka Matsuo. Earthquake shakes twitter users: real-time event detection by social sensors. In *Proceedings of the 19th international conference on World wide web*, pages 851–860. ACM, 2010.

[190] Rosamaria Salvatori, Paolo Plini, Marco Giusto, and et al. Snow cover monitoring with images from digital camera systems. *Italian Journal of Remote Sensing*, 43:137–145, 2011.

[191] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018.

[192] Carlos GR Santos, Tiago Araújo, Paulo R Chagas, Nelson Neto, and Bianchi S Meiguins. Recognizing and exploring azulejos on historic buildings' facades by combining computer vision and geolocation in mobile augmented reality applications. *Journal of Mobile Multimedia*, 13(1-2):57–74, 2017.

[193] Olivier Saurer, Georges Baatz, Kevin Köser, Marc Pollefeys, et al. Image based geo-localization in the alps. *International Journal of Computer Vision*, 116(3):213–225, 2016.

[194] Svend-Jonas Schelhorn, Benjamin Herfort, Richard Leiner, Alexander Zipf, and João Porto De Albuquerque. Identifying elements at risk from openstreetmap: The case of flooding. In *ISCRAM*, 2014.

[195] Dominik Scherer, Andreas Müller, and Sven Behnke. Evaluation of pooling operations in convolutional architectures for object recognition. In *International conference on artificial neural networks*, pages 92–101. Springer, 2010.

[196] E Schnebele, G Cervone, and N Waters. Road assessment after flood events using non-authoritative data. *Natural Hazards and Earth System Science*, 2014.

[197] Joachim Scholz and Andrew N Smith. Augmented reality: Designing immersive experiences that maximize consumer engagement. *Business Horizons*, 59(2):149–161, 2016.

[198] Abhishek Sehgal and Nasser Kehtarnavaz. Guidelines and benchmarks for deployment of deep learning models on smartphones as real-time apps. *arXiv preprint arXiv:1901.02144*, 2019.

[199] Victor S Sheng, Foster Provost, and Panagiotis G Ipeirotis. Get another label? improving data quality and data mining using multiple, noisy labelers. In *Proceedings of the 14th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 614–622. ACM, 2008.

[200] Shaohuai Shi, Qiang Wang, Pengfei Xu, and Xiaowen Chu. Benchmarking state-of-the-art deep learning software tools. In *7th International Conference on Cloud Computing and Big Data (CCBD)*, pages 99–104. IEEE, 2016.

[201] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

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

[203] Robert Simpson, Kevin R Page, and David De Roure. Zooniverse: observing the world's largest citizen science platform. In *Proceedings of the 23rd international conference on world wide web*, pages 1049–1054. ACM, 2014.

[204] Rion Snow, Brendan O'Connor, Daniel Jurafsky, and Andrew Y Ng. Cheap and fast—but is it good?: evaluating non-expert annotations for natural language tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 254–263. Association for Computational Linguistics, 2008.

[205] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[206] Xing Su, Hanghang Tong, and Ping Ji. Activity recognition with smartphone sensors. *Tsinghua Science and Technology*, 19(3):235–249, 2014.

[207] Brian L Sullivan, Christopher L Wood, Marshall J Iliff, Rick E Bonney, Daniel Fink, and Steve Kelling. ebird: A citizen-based bird observation network in the biological sciences. *Biological Conservation*, 142(10):2282–2292, 2009.

[208] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[209] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[210] Mingxing Tan, Bo Chen, Ruoming Pang, Vijay Vasudevan, and Quoc V Le. Mnasnet: Platform-aware neural architecture search for mobile. *arXiv preprint arXiv:1807.11626*, 2018.

[211] Giovanni Taverriti, Stefano Lombini, Lorenzo Seidenari, Marco Bertini, and Alberto Del Bimbo. Real-time wearable computer vision system for improved museum experience. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 703–704. ACM, 2016.

[212] Ben Taylor, Vicent Sanz Marco, Willy Wolff, Yehia Elkhatib, and Zheng Wang. Adaptive deep learning model selection on embedded systems. In *Proceedings of the 19th ACM SIGPLAN/SIGBED International Conference on Languages, Compilers, and Tools for Embedded Systems*, pages 31–43. ACM, 2018.

[213] Arvind Thiagarajan, Lenin Ravindranath, Katrina LaCurts, Samuel Madden, Hari Balakrishnan, Sivan Toledo, and Jakob Eriksson. Vtrack: accurate, energy-aware road traffic delay estimation using mobile phones. In *Proceedings of the 7th ACM conference on embedded networked sensor systems*, pages 85–98. ACM, 2009.

[214] Yoshitaka Tokusho and Steven Feiner. Prototyping an outdoor mobile augmented reality street view application. In *Proceedings of ISMAR Workshop on Outdoor Mixed and Augmented Reality*, volume 2. Citeseer, 2009.

[215] Rocio Nahime Torres, Darian Frajberg, Piero Fraternali, and Sergio Luis Herrera Gonzales. Crowdsourcing landforms for open gis enrichment. In *2018 IEEE 5th International Conference on Data Science and Advanced Analytics (DSAA)*, pages 604–611. IEEE, 2018.

[216] Grant Van Horn, Steve Branson, Ryan Farrell, Scott Haber, Jessie Barry, Panos Ipeirotis, Pietro Perona, and Serge Belongie. Building a bird recognition app and large scale dataset with citizen scientists: The fine print in fine-grained dataset collection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 595–604, 2015.

[217] Grant Van Horn, Oisin Mac Aodha, Yang Song, Yin Cui, Chen Sun, Alex Shepard, Hartwig Adam, Pietro Perona, and Serge Belongie. The inaturalist species classification and detection dataset. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 8769–8778, 2018.

[218] Vaninha Vieira, Konstantin Holl, and Michael Hassel. A context simulator as testing support for mobile apps. In Roger L. Wainwright, Juan Manuel Corchado, Alessio Bechini, and Jiman Hong, editors, *Proceedings of the 30th Annual ACM Symposium on Applied Computing, Salamanca, Spain, April 13-17, 2015*, pages 535–541. ACM, 2015.

[219] Ji Wang, Bokai Cao, Philip Yu, Lichao Sun, Weidong Bao, and Xiaomin Zhu. Deep learning towards mobile applications. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1385–1393. IEEE, 2018.

[220] Jingya Wang, Mohammed Korayem, and David J Crandall. Observing the natural world with flickr. In *Computer Vision Workshops (ICCVW), 2013 IEEE International Conference on*, pages 452–459. IEEE, 2013.

[221] Ruohui Wang. Edge detection using convolutional neural network. In *Int. Symposium on Neural Networks*, pages 12–20. Springer, 2016.

[222] Yiwei Wang, Nicole Kaplan, Greg Newman, and Russell Scarpino. Citsci. org: A new model for managing, documenting, and sharing citizen science data. *PLoS biology*, 13(10):e1002280, 2015.

[223] Tobias Weyand, Ilya Kostrikov, and James Philbin. Planet-photo geolocation with convolutional neural networks. In *European Conference on Computer Vision*, pages 37–55. Springer, 2016.

[224] Lee J White. Regression testing of gui event interactions. In *Software Maintenance 1996, Proceedings., International Conference on*, pages 350–358. IEEE, 1996.

[225] Jacob Whitehill, Ting-fan Wu, Jacob Bergsma, Javier R Movellan, and Paul L Ruvolo. Whose vote should count more: Optimal integration of labels from labelers of unknown expertise. In *Advances in neural information processing systems*, pages 2035–2043, 2009.

[226] Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. Fbnet: Hardware-aware efficient convnet design via differentiable neural architecture search. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019.

[227] Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

[228] Saining Xie and Zhuowen Tu. Holistically-nested edge detection. In *Proceedings of the IEEE international conference on computer vision*, pages 1395–1403, 2015.

[229] Bing Xu, Naiyan Wang, Tianqi Chen, and Mu Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[230] Haipeng Zhang, Mohammed Korayem, David J Crandall, and Gretchen LeBuhn. Mining photo-sharing websites to study ecological phenomena. In *Proceedings of the 21st international conference on World Wide Web*, pages 749–758. ACM, 2012.

[231] Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 6848–6856, 2018.

[232] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *IEEE Conf. on Computer Vision and Pattern Recognition (CVPR)*, pages 2881–2890, 2017.

[233] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen. Incremental network quantization: Towards lossless cnns with low-precision weights. *arXiv preprint arXiv:1702.03044*, 2017.

[234] Barbara Zitova and Jan Flusser. Image registration methods: a survey. *Image and Vision Computing*, 21(11):977 – 1000, 2003.

[235] Matthew Zook, Mark Graham, Taylor Shelton, and Sean Gorman. Volunteered geographic information and crowdsourcing disaster relief: a case study of the haitian earthquake. *Available at SSRN 2216649*, 2010.

[236] Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.

APPENDIX A

Comprehensive inference time experimental results

This appendix presents comprehensive tables with more detailed information about the experiment discussed in Section 6.3.3 to analyze the performance of TensorFlow Lite and PolimiDL inference execution frameworks. The following tables include the values for each of the three evaluation iterations (1-2-3) executed on each combination of model, device, and configuration; their average (AVG); and the percentage difference (DIFF) of PolimiDL w.r.t. TensorFlow Lite. Moreover, the Sample Standard Deviation (S) is included to provide an indication on the stability of the experimental results. Positive results for PolimiDL are highlighted in green and negative results in red.

Table A.1 reports the experimental results for *PL Original Model* and extends Table 6.4; Table A.2 reports the experimental results for *PL Optmized Model v1* and extends Table 6.5; and Table A.3 reports the experimental results for *MobileNet* and extends Table 6.6.

	TensorFlow Lite (ms)													PolimiDL (ms)										
	Min(4, Threads)					Max(1, Threads-1)					All Threads					Min(4, Threads)					Max(1, Threads-1)					All Threads				
Device	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**
Asus Zenfone 2	1354	1352	1352	1.15	1352.67	1674	1672	1672	1.15	1672.67	1355	1353	1351	2.00	1353.00	938	937	933	2.65	936.00	**-30.80**	1140	1137	1137	1.73	1138.00	**-31.96**	935	943	932	5.69	936.67	**-30.77**
Google Pixel	210	206	207	2.08	207.67	260	254	252	4.16	255.33	214	209	208	3.21	210.33	144	145	146	1.00	145.00	**-30.18**	168	174	171	3.00	171.00	**-33.03**	146	144	145	1.00	145.00	**-31.06**
LG G5 SE	417	416	423	3.79	418.67	282	300	288	9.17	290.00	270	277	271	3.79	272.67	276	274	269	3.61	273.00	**-34.79**	208	210	209	1.00	209.00	**-27.93**	204	201	196	4.04	200.33	**-26.53**
LG Nexus 5X	424	424	423	0.58	423.67	371	369	371	1.15	370.33	335	340	334	3.21	336.33	434	433	430	2.08	432.33	**2.05**	343	343	341	1.15	342.33	**-7.56**	287	279	281	4.16	282.33	**-16.06**
Motorola Nexus 6	335	336	339	2.08	336.67	507	520	489	15.57	505.33	339	334	340	3.21	337.67	169	169	169	0.00	169.00	**-49.80**	217	215	215	1.15	215.67	**-57.32**	170	167	168	1.53	168.33	**-50.15**
One Plus 6T	176	175	177	1.00	176.00	143	145	145	1.15	144.33	142	146	148	3.06	145.33	103	104	105	1.00	104.00	**-40.91**	90	91	92	1.00	91.00	**-36.95**	89	89	89	0.00	89.00	**-38.76**
Average	-	-	-	**1.78**	**485.89**	-	-	-	**5.39**	**539.67**	-	-	-	**3.08**	**442.56**	-	-	-	**1.72**	**343.22**	**-30.74**	-	-	-	**1.51**	**361.17**	**-32.46**	-	-	-	**2.74**	**303.61**	**-32.22**

Table A.1: *Comprehensive experimental results of PL Original Model.*

	TensorFlow Lite (ms)															PolimiDL (ms)																
	Min(4, Threads)					Max(1, Threads-1)					All Threads					Min(4, Threads)						Max(1, Threads-1)						All Threads				
Device	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**
Asus Zenfone 2	741	740	741	0.58	740.67	804	814	805	5.51	807.67	742	742	746	2.31	743.33	167	165	166	1.00	166.00	**-77.59**	179	180	179	0.58	179.33	**-77.80**	166	171	166	2.89	167.67	**-77.44**
Google Pixel	82	82	82	0.00	82.00	95	96	94	1.00	95.00	85	82	81	2.08	82.67	30	30	30	0.00	30.00	**-63.41**	34	35	37	1.53	35.33	**-62.81**	31	31	31	0.00	31.00	**-62.50**
LG G5 SE	165	184	208	21.55	185.67	137	141	137	2.31	138.33	136	135	143	4.36	138.00	88	97	98	5.51	94.33	**-49.19**	66	68	70	2.00	68.00	**-50.84**	73	68	71	2.52	70.67	**-48.79**
LG Nexus 5X	204	205	204	0.58	204.33	193	194	192	1.00	193.00	180	182	181	1.00	181.00	85	85	84	0.58	84.67	**-58.56**	82	80	79	1.53	80.33	**-58.38**	79	76	76	1.73	77.00	**-57.46**
Motorola Nexus 6	130	148	143	9.29	140.33	216	232	229	8.50	225.67	131	139	137	4.16	135.67	57	52	48	4.51	52.33	**-62.71**	68	66	64	2.00	66.00	**-70.75**	52	49	46	3.00	49.00	**-63.88**
One Plus 6T	67	66	67	0.58	66.67	69	68	69	0.58	68.67	66	67	66	0.58	66.33	22	22	22	0.00	22.00	**-67.00**	23	23	22	0.58	22.67	**-66.99**	22	22	23	0.58	22.33	**-66.33**
Average	**-**	**-**	**-**	**5.43**	**236.61**	**-**	**-**	**-**	**3.15**	**254.72**	**-**	**-**	**-**	**2.42**	**224.50**	-	**-**	**-**	**1.93**	**74.89**	**-63.08**	**-**	**-**	**-**	**1.37**	**75.28**	**-64.59**	**-**	**-**	**-**	**1.79**	**69.61**	**-62.73**

Table A.2: *Comprehensive experimental results of PL Optmized Model v1.*

	TensorFlow Lite (ms)												PolimiDL (ms)											
	Min(4, Threads)					Max(1, Threads-1)					All Threads												All Threads	
Device	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**	**1**	**2**	**3**	**S**	**AVG**	**DIFF(%)**
Asus Zenfone 2	734	735	733	1.00	734.00	775	774	777	1.53	775.33	732	734	734	1.15	733.33	372	371	370	1.00	371.00	**-49.46**	378	378	376	1.15	377.33	**-51.33**	377	372	374	2.52	374.33	**-48.95**
Google Pixel	76	76	75	0.58	75.67	84	82	81	1.53	82.33	77	75	79	2.00	77.00	73	76	73	1.73	74.00	**-2.20**	82	83	83	0.58	82.67	**0.40**	73	75	73	1.15	73.67	**-4.33**
LG G5 SE	257	270	264	6.51	263.67	273	279	272	3.79	274.67	274	277	276	1.53	275.67	274	279	277	2.52	276.67	**4.93**	257	257	263	3.46	259.00	**-5.70**	253	269	247	11.37	256.33	**-7.01**
LG Nexus 5X	216	218	218	1.15	217.33	226	225	224	1.00	225.00	222	224	224	1.15	223.33	222	222	223	0.58	222.33	**2.30**	239	236	228	5.69	234.33	**4.15**	226	226	226	0.00	226.00	**1.19**
Motorola Nexus 6	229	220	224	4.51	224.33	296	300	299	2.08	298.33	225	227	231	3.06	227.67	205	207	199	4.16	203.67	**-9.21**	180	171	177	4.58	176.00	**-41.01**	193	132	165	30.53	163.33	**-28.26**
One Plus 6T	58	56	56	1.15	56.67	57	57	56	0.58	56.67	59	57	57	1.15	57.67	49	50	50	0.58	49.67	**-12.35**	51	52	52	0.58	51.67	**-8.82**	54	51	54	1.73	53.00	**-8.09**
Average	-	-	-	**2.48**	**261.94**	-	-	-	**1.75**	**285.39**	-	-	-	**1.67**	**265.78**	-	-	-	**1.76**	**199.56**	**-11.00**	-	-	-	**2.67**	**196.83**	**-17.05**	-	-	-	**7.88**	**191.11**	**-15.91**

Table A.3: *Comprehensive experimental results of MobileNet model.*