Deep Learning architectures for onboard satellite image analysis

Gaétan Bahl

UNIVERSITÉ CÔTE D'AZUR

ÉCOLE DOCTORALE
SCIENCES ET TECHNOLOGIES DE L'INFORMATION ET DE LA COMMUNICATION

THÈSE DE DOCTORAT

Architectures Deep Learning pour l'analyse d'images satellite embarquée

Gaétan Bahl

Équipe TITANE, Inria Sophia-Antipolis Méditerranée

Présentée en vue de l'obtention du grade de docteur en Informatique d'Université Côte d'Azur

Dirigée par : Florent Lafarge

Soutenue le : 2 Juin 2022

Devant le jury, composé de :

Pierre Alliez	Directeur de recherche	Inria
Adrien Girard	Ingénieur	IRT Saint Exupéry
Christian Heipke	Professeur	LUH
Florent Lafarge	Directeur de recherche	Inria
Sébastien Lefèvre	Professeur	IRISA
Yuliya Tarabalka	Chargé de recherche	LuxCarta

informatics mathematics

IRT SAINT EXUPÉRY

Architectures Deep Learning pour l'analyse d'images satellite embarquée

Deep Learning architectures for onboard satellite image analysis

Jury :

Rapporteurs

Christian Heipke	Professeur	Leibniz Universität Hannover
Sébastien Lefèvre	Professeur	IRISA, Université Bretagne Sud

Examinateurs

Pierre Alliez	Directeur de recherche	Inria Sophia-Antipolis
Adrien Girard	Ingénieur	IRT Saint Exupéry
Florent Lafarge	Directeur de recherche	Inria Sophia-Antipolis
Yuliya Tarabalka	Chargé de recherche	LuxCarta

Acknowledgements

I first would like to express my gratitude towards the members of the committee for agreeing to review this thesis and for their constructive comments.

Second, I would like to thank my supervisor, Florent Lafarge, for his valuable advice during the preparation of this thesis. I think we had a good time, all things considered, and I hope your next students will achieve what I never could: beating you at King of Fighters. I also would like to thank Pierre Alliez, head of the TITANE team at Inria, for welcoming me into the team and allowing me to teach by his side. Thanks to all the other team members and people from the Byron building for being amazing co-workers and always being up for a game or two.

This work was funded by the CIAR project at IRT Saint-Exupéry. Thanks to all members of the project, and in particular to the managers Frédéric Ferésin and Adrien Girard for allowing me to have academic freedom, and to Lionel Daniel for the fruitful discussions and help. I am grateful to the OPAL infrastructure from Université Côte d'Azur for providing resources and support. This work was granted access to the HPC resources of IDRIS under the allocation 2020-AD011011311R2 made by GENCI.

I believe all past experiences, good or bad, take part in shaping who we are. There are many people who's actions helped me along the way, and without whom my life would be very different. Former teachers, supervisors, managers, colleagues and classmates: I am grateful for everything that you have done.

Thanks to all my friends for supporting me, listening to me and giving me strength. I have not been the greatest friend these past few years. Thanks for sticking by my side anyway, I'll make sure to get better. Thanks in particular to Mehdi Bahri, you have been a great mentor over the years and it was a pleasure to finally work with you.

Je remercie ma famille pour son support pendant toutes ces années, et pour avoir dès le plus jeune âge éveillé ma curiosité scientifique. Vous ne compreniez pas forcément pourquoi je passais autant de temps devant l'ordinateur, mais vous m'avez laissé faire quand même et je vous en remercie.

Last but not least, I would like to thank my girlfriend, Anaïs, without whom I probably would not have finished – or even started – this project. I would support you in the same way if you ever needed it. I am lucky to have you in my life.

Abstract

The recent advances in high-resolution Earth observation satellites and the reduction in revisit times introduced by the creation of constellations of satellites has led to the daily creation of large amounts of image data (hundreds of TeraBytes per day). Simultaneously, the popularization of Deep Learning techniques allowed the development of architectures capable of extracting semantic content from images. While these algorithms usually require the use of powerful hardware, low-power AI inference accelerators have recently been developed and have the potential to be used in the next generations of satellites, thus opening the possibility of onboard analysis of satellite imagery. By extracting the information of interest from satellite images directly onboard, a substantial reduction in bandwidth, storage and memory usage can be achieved. Current and future applications, such as disaster response, precision agriculture and climate monitoring, would benefit from a lower processing latency and even real-time alerts.

In this thesis, our goal is two-fold: On the one hand, we design efficient Deep Learning architectures that are able to run on low-power edge devices, such as satellites or drones, while retaining a sufficient accuracy. On the other hand, we design our algorithms while keeping in mind the importance of having a compact output that can be efficiently computed, stored, transmitted to the ground or other satellites within a constellation.

First, by using depth-wise separable convolutions and convolutional recurrent neural networks, we design efficient semantic segmentation neural networks with a low number of parameters and a low memory usage. We apply these architectures to cloud and forest segmentation in satellite images. We also specifically design an architecture for cloud segmentation on the FPGA of OPS-SAT, a satellite launched by ESA in 2019, and perform onboard experiments remotely. Second, we develop an instance segmentation architecture for the regression of smooth contours based on the Fourier coefficient representation, which allows detected object shapes to be stored and transmitted efficiently. We evaluate the performance of our method on a variety of low-power computing devices. Finally, we propose a road graph extraction architecture based on a combination of fully convolutional and graph neural networks. We show that our method is significantly faster than competing methods, while retaining a good accuracy.

Keywords: Deep learning, remote sensing, satellite imagery, semantic segmentation, instance segmentation, graph extraction, edge computing

Résumé

Les progrès des satellites d'observation de la Terre à haute résolution et la réduction des temps de revisite introduite par la création de constellations de satellites ont conduit à la création quotidienne de grandes quantités d'images (des centaines de Teraoctets par jour). Simultanément, la popularisation des techniques de Deep Learning a permis le développement d'architectures capables d'extraire le contenu sémantique des images. Bien que ces algorithmes nécessitent généralement l'utilisation de matériel puissant, des accélérateurs d'inférence IA de faible puissance ont récemment été développés et ont le potentiel d'être utilisés dans les prochaines générations de satellites, ouvrant ainsi la possibilité d'une analyse embarquée des images satellite. En extrayant les informations intéressantes des images satellite directement à bord, il est possible de réduire considérablement l'utilisation de la bande passante, du stockage et de la mémoire. Les applications actuelles et futures, telles que la réponse aux catastrophes, l'agriculture de précision et la surveillance du climat, bénéficieraient d'une latence de traitement plus faible, voire d'alertes en temps réel.

Dans cette thèse, notre objectif est double : D'une part, nous concevons des architectures de Deep Learning efficaces, capables de fonctionner sur des périphériques de faible puissance, tels que des satellites ou des drones, tout en conservant une précision suffisante. D'autre part, nous concevons nos algorithmes en gardant à l'esprit l'importance d'avoir une sortie compacte qui peut être efficacement calculée, stockée, transmise au sol ou à d'autres satellites dans une constellation.

Tout d'abord, en utilisant des convolutions séparables en profondeur et des réseaux neuronaux récurrents convolutionnels, nous concevons des réseaux neuronaux de segmentation sémantique efficaces avec un faible nombre de paramètres et une faible utilisation de la mémoire. Nous appliquons ces architectures à la segmentation des nuages et des forêts dans les images satellites. Nous concevons également une architecture spécifique pour la segmentation des nuages sur le FPGA d'OPS-SAT, un satellite lancé par l'ESA en 2019, et réalisons des expériences à bord à distance. Deuxièmement, nous développons une architecture de segmentation d'instance pour la régression de contours lisses basée sur une représentation

à coefficients de Fourier, qui permet de stocker et de transmettre efficacement les formes des objets détectés. Nous évaluons la performance de notre méthode sur une variété de dispositifs informatiques à faible puissance. Enfin, nous proposons une architecture d'extraction de graphes routiers basée sur une combinaison de *Fully Convolutional Networks* et de *Graph Neural Networks*. Nous montrons que notre méthode est nettement plus rapide que les méthodes concurrentes, tout en conservant une bonne précision.

Mots-clefs: Apprentissage profond, télédétection, imagerie satellite, segmentation sémantique, segmentation d'instance, extraction de graphes, traitement à bord

Contents

Chapter 1

Introduction

"L'uomo verrà portato dalla sua creazione.
Come gli uccelli, verso il cielo...
Riempendo l'universo di stupore e gloria."
— Christopher Tin, *Sogno di Volare*

In this first chapter, we start by introducing the context and motivation behind this thesis. We also give a general overview of state-of-the-art methods, with their advantages, drawbacks, and challenges. Finally, we propose an outline of the contributions of this work.

1.1 Context and motivation

Being able to easily and accurately monitor large areas of land and react to changes in a timely manner has always been a necessity, for survival, protection, and exploration. This practice probably started millions of years ago by climbing atop of trees to scout for food or predators. In the recent millenia, humans have built tall buildings, towers and castles on top of hills and mountains. Similarly, the goal was to react to possible invaders, and more generally monitor the state of the land and act appropriately. At the beginning of the 20th century, mankind took to the skies with the creation of the first airplanes. During the wars that followed, they were used for aerial reconnaissance, along with balloons and zeppelins. Again, the objective was to provide vital information with the lowest possible latency.

The launch of the first satellite in 1957 kicked off the *Space Race*, which led to the creation of the first Earth Observation (EO) satellites in the 1960's. In just a few decades, the number of EO satellites in orbit has grown to more than 900. These satellites feature varying equipments, such as radar imagers, or multi-spectral/hyperspectral imagers, and are thus used for different purposes, such as

meteorology, natural disaster monitoring or defense. Most of the satellites feature some kind of imaging system, the most common one being sun reflectance-based imaging, where a sensor measures the light from the sun being reflected by the Earth's surface.

Over the years, the capabilities of imaging satellites have grown as much as their number. Sensors got more precise and constellations of satellites have lowered the revisit time substantially. In consequence, the sheer volume of data created by EO satellites is hard to even comprehend. Back in 2008, we were already talking about a total of more than 10 Terabytes of daily acquisitions [TGH08]. Today, *Planet* alone produces more than 25 Terabytes of data every day with its constellations of small satellites and in the near future, the *Pléiades Neo* constellation will be able to produce 40 Terabytes of image data per day.

As launch costs keep diminishing and the number of satellites keeps growing, these enormous quantities of data will keep growing as well, which poses a number of challenges. The acquisition of images is only one step. These images have to be processed onboard, transmitted to the ground, stored in data centers, and exploited using powerful servers. This requires huge amounts of bandwidth, storage space and processing power. Moreover, some processes are not automated yet. The annotation of images for the creation of maps, for example, is still largely a manual task. The analysis of images for disaster response is also done manually, which imposes a latency bottleneck and thus lengthens reaction time.

In parallel to the development of satellites and imaging sensors, great strides have been made in both the field of artificial intelligence and the field of edge computing. Since the success of AlexNet in the ImageNet image classification challenge [KSH12], Convolutional Neural Networks (CNNs) have been used for all kinds of computer vision tasks, such as object detection and semantic segmentation, and have virtually taken the world of computer vision by storm. While the power requirements to use these algorithms were originally huge (*i.e.* machines with multiple powerful *Graphics Processing Units* or GPUs), they are nowadays used *on the edge* in a variety of applications, including smartphones, home assistants, UAVs and cars, thanks to low-power architectures and various techniques used to port them to edge computing hardware.

With the creation of low-power hardware capable of running CNN inference, such as the Nvidia Jetson family of devices, it is only a matter of time before future satellites also feature AI inference accelerators. In fact, some CubeSat missions have already taken a step in that direction. For example, PhiSat-1 launched in September 2020 with an Intel Movidius Myriad 2 *Vision Processing Unit*. These new processing capabilities will allow to intelligently process the data on-board in order to save time, bandwidth, and storage. Indeed, not all the information captured in satellite imagery is useful for every application, and most of it might

not even need to be transferred back to Earth. A simple example is clouds, which cover more than 50% of the planet's sky at any given time, and are useless in a lot of applications. The on-board extraction of the final useful product from images is also a possibility, where the satellites would only transmit the required information, such as land cover, object positions, or road networks. Finally, fast on-board processing with machine learning algorithms could lead to real-time alerts for natural disaster response, agricultural monitoring, or surveillance.

This is where the work presented in this thesis takes place. Our goal is two-fold:

- We design compact and efficient neural network architectures that are able to run on the kind of low-power hardware that will be featured on-board satellites in the future.
- We also require these architectures to generate compact *outputs*, in the form of compressible images and masks, sparse graphs or representations, in order to save bandwidth during transmission to the ground, save storage space both on-board and on the ground, and be memory-friendly during inference.

1.1.1 Satellite imagery

In this section, we present some of the particularities of satellite imagery compared to other types of image acquisition devices.

The invention of CCD and CMOS sensors and their use in cameras has allowed a large number of consumer devices to gain the ability to take pictures and videos. The capabilities of each device can vary greatly: indeed, sensor type and size, ISO range, focal length and zoom capabilities all contribute to the creation of widely different pictures from one camera to the other. Even using the same camera model, two different users might take very different pictures. *E.g.* one might be more of a landscape photographer, and the other more of a portrait photographer. Exposure, orientation, and subject will also vary greatly. All of these hardware-induced and user-induced variables create a large variety of photos, and thus require different algorithms for each purpose.

On the contrary, satellite imagery is actually simpler in that regard. The sensors onboard each satellite will never change during its lifespan, which eliminates a large number of variables. Moreover, satellites operate at a fixed altitude and generally look towards the ground, this means that the *Ground Sampling Distance* (GSD), which is the size of one pixel on the ground, will rarely change either. Geostationary satellites even survey a fixed area of the planet, which means position and orientation never change in that case. Since satellites operate at a high altitude, the effects of perspective are also reduced. These simplifications allow the design and training of algorithms on satellite data to actually be somewhat easier in some aspects than for more general purpose applications.

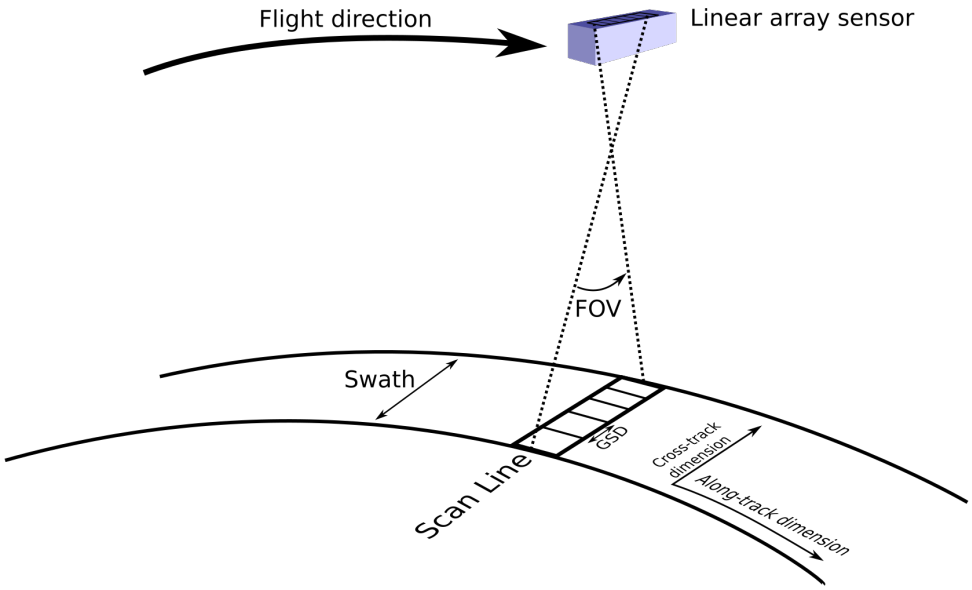


Figure 1.1: Acquisition of an image using a *push broom* sensor.

Let us go over some commonly used terminology:

Sensors: The word "sensor" refers to the actual measurement devices onboard a satellite. In the case of optical imagery, a sensor is an array of passive radiometers that measure the intensity of electromagnetic radiation from the Sun reflected by the Earth surface within one or several bands. Imagery sensors come into two categories:

- *Frame* sensors are arranged in a matrix, similarly to the ones present in any regular camera. Aerial imagery from planes or *Unmanned Aerial Vehicles* (UAVs) also generally uses *frame* sensors. This type of sensor is often featured on small satellites, such as CubeSats and NanoSats. With these sensors, the whole image is acquired at once, *i.e.* the two dimensions of the image are created by the two dimensions of the sensor.

- *Scanner* sensors include *Push broom* (shown on Figure 1.1) and *Whisk broom* sensors. With *Push broom* sensors, the detectors are arranged in a single or multiple line array, which creates the first dimension of the image (also called *cross-track* dimension). In *Whisk broom* sensors, on the other hand, a single photosite is present, and the first dimension of the image is created by the rotation of a mirror. In both cases, the second dimension of the image, also called *along-track* dimension, is created by the movement of the satellite

platform, just like a document scanner. Thus, large output images can be generated by letting the acquisition run for a long time. The size of the cross-track dimension projected on the ground is called the *swath* of the sensor, measured in meters. *Push broom* sensors are the most common type of sensors featured in high-end optical imagery satellites, such as Pléiades and Landsat-8.

Most of the methods presented in this thesis can be applied to images from both *frame* and *scanner* types of sensors. However, we will see in Chapter 3 that it is possible to design neural network architectures that are specifically tailored towards *scanner* sensors.

Temporal resolution, or *revisit time* refers to the amount of time that passes between two *revistits* of a satellite over a certain location. The revisit time is usually measured in days. For example, Landsat-8 can visit a single location once every 16 days. By using multiple satellites, recent constellations such as PlanetScope allow reduced revisit times and daily coverage of the whole planet. Future platforms such as Pléiades Neo will even allow intra-day monitoring, with a revisit time of 12 hours. A high temporal resolution will allow the accurate monitoring of fast evolving phenomena. However, it will also generate larger quantities of data.

Spectral bands Radiometry sensors feature one or several *bands* or *channels*. Each band of a sensor corresponds to a specific range of wavelengths, to which it responds. For example, Band 4 on Landsat-8 responds to wavelengths in the 0.64 to 0.67µm range, which corresponds to the color red. Most cameras featured in consumer electronics only feature three bands, responding to Red, Green and Blue. Earth Observation satellites, however, often feature considerably more spectral bands, each with specific uses. For example, Sentinel-2 features 12 bands, which can be used in different combinations to highlight vegetation, moisture or geologic features.

A *Panchromatic band* (PAN), is what we would more commmonly refer to as a "black and white" band. Such a band responds to a wide range of wavelengths in the visible spectrum. The panchromatic band of a sensor often has a better spatial resolution (or GSD) than other bands. This allows the creation of images combining the details of the panchromatic band, and the colors of other bands, through a process called *pan-sharpening*.

RGB sensors use three bands to produce images with natural colors, as we, humans, would see them. They are often combined with a fourth band, called *NIR*, which responds to *Near InfraRed*. *RGBN* sensors, *i.e.* RGB+Nir, are particularly useful to create *false color* images that highlight vegetation, as shown in Figure 1.2.

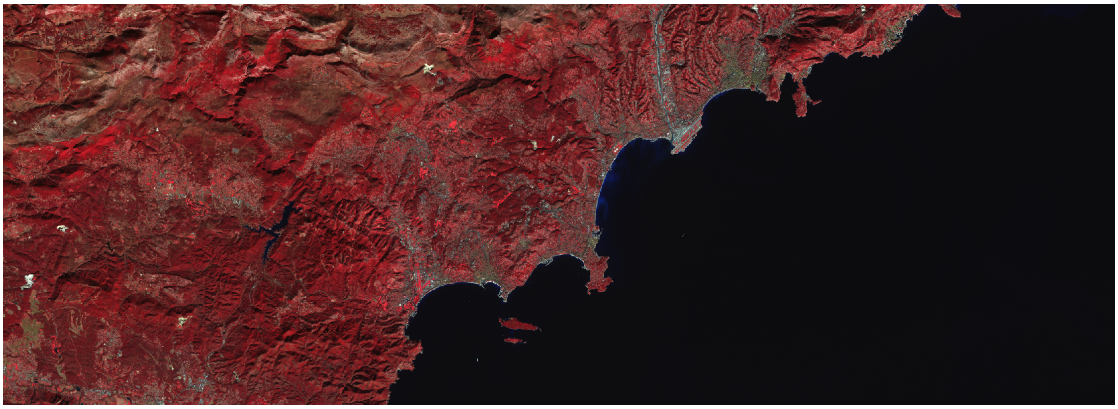


Figure 1.2: False color image of *Côte d'Azur, France* created with bands 8 (NIR), 4 (red) and 3 (green) of Sentinel-2, in order to highlight vegetation. Data source: USGS.

Multispectral sensors (MS) are sensors which typically feature 3 to around 10 bands. Landsat-8 and Sentinel-2 are examples of multispectral sensors. *Hyperspectral* sensors can feature hundreds of different bands. The *radiometric* resolution of a sensor is the number of bits used to represent each pixel of each band, while the *spectral* resolution corresponds to the number of different bands, and their width in terms of wavelength. Some objects that might be hard to detect in the visible spectrum (*e.g.* some aquatic species) could be easily highlighted using multispectral or hyperspectral sensors.

In this work, we focus primarily on RGB images, as they are the most commonly available. However, all the architectures we developed can also be used on images with more than three channels.

Ground Sampling Distance, or *GSD* is the spatial resolution of a sensor, and is measured in meters. It corresponds to the size of one pixel projected on the ground. Different sensors or spectral bands might have different GSDs, even on the same satellite platform. GSDs in the range of 1 to 10 meters are very common nowadays, with *Very High Resolution* (VHR) sensors reaching resolutions in the tens of centimeters. Figure 1.3 shows the difference in details between GSD values of some popular satellite platforms. The spatial resolution of satellites will continue to improve in the future. As of today, it has not reached what is possible to do with aerial imaging (*i.e.* planes, UAVs).

While having more resolution and thus more information at our disposal is interesting for some applications, it can be unnecessary or might even be detrimental in other applications. For example, a GSD of around 2 meters might be enough to discern most roads in a road extraction application, and working at a higher

(a) Landsat-8, GSD=30m (b) Sentinel-2, GSD=10m (c) Pléiades-1A, GSD=2m

Figure 1.3: Part of *Cap d'Antibes*, as seen by Earth Observation satellites with differing Ground Sampling Distances (GSD). Data source: USGS, CNES.

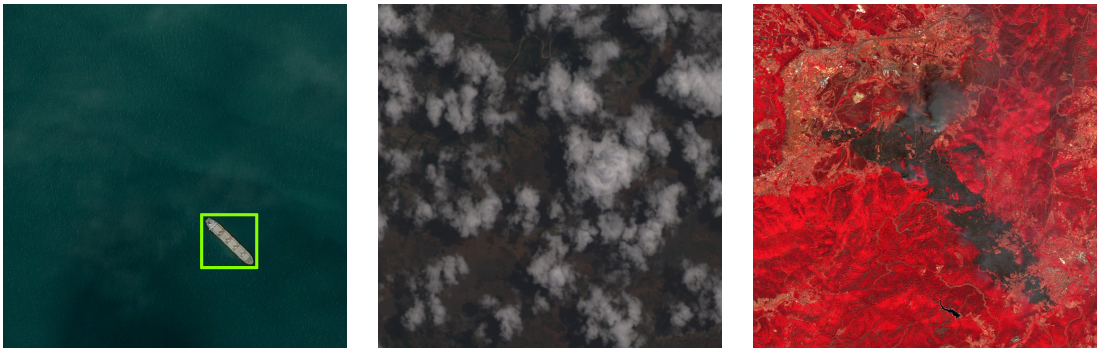
resolution might introduce more noise in the form of unnecessary details, like cars or pedestrians. Naturally, it is always possible to virtually reduce the resolution of input images by resampling them, as we will see in Chapter 5.

1.1.2 On-board processing

In this section, we delve into the multiple potential benefits of onboard processing of satellite images.

Remote sensing satellites operate in a store-and-forward fashion. Images are acquired and stored onboard until the satellite flies over a ground station, at which point the data is offloaded. Naturally, this transmission can only happen when a ground station is in view. This raises three main issues: First, onboard *storage space* is crucial, since it acts as an image buffer. While storage costs have diminished in recent years, it is not possible to upgrade it after launch. Second, *bandwidth* requirements are also important. Indeed, the data needs to be transmitted in a minimum of passes over ground stations in order to avoid any down-time. Third, the *latency* induced by this process is very high. Users not only have to wait for image acquisition and transmission, but also for compute-intensive image processing on the ground.

Storage space is not only a limitation onboard, but also on the ground. Satellite images are stored in data centers and in most cases replicated accross multiple machines and hard drives, which have a significant footprint, both in terms of space and in terms of energy. As the resolution of sensors keeps increasing and the number of satellites grows, the strain on storage and bandwidth will only grow further. Moreover, the development of constellations comprising dozens or even hundreds of satellites will also demand more ground stations and more bandwidth.



(a) Single ship in image (b) Cloud-covered image (c) Forest fire

Figure 1.4: Images from (a) the Airbus Ship Detection Challenge, (b) 38-Cloud dataset [MS19] and (c) Sentinel-2 false color image of a 2021 forest fire in the Var region of France. The relevant information in these images can be expressed using only a few bytes.

The latency issues have led to the creation of data relay satellites, such as the *European Data Relay Satellite System* (EDRS), which can collect data from Earth Observation satellites and transmit it to ground stations. While this is particularly useful in time-critical services (*i.e.* natural disasters, rescue), such systems take a huge effort and a long time to be put in place. They will also have a limited bandwidth and thus will only be able to handle a limited amount of data at a time.

Storage, bandwidth, and latency limitations are in part due to the fact that full resolution images are used throughout the whole acquisition and transmission process. A single Sentinel-2 scene, for example can have more than 700MB. However, the relevant information contained in images can often be expressed using very limited quantities of data. Figure 1.4 shows three examples of that:

- The position of the ship in the first image can be summed up using two floating point coordinates, and its bounding box in the image can be expressed with four integers.
- The second image is largely occluded by clouds. If the percentage of occlusion is too high, this image can be discarded directly and never be transmitted to the ground. However, if this quantity of clouds is interesting, it can be expressed with a single integer. The image can also be compressed by simply deleting all the unwanted cloud pixels.
- In the third image, we observe that a part of the forest (in brown/black) has already burned, while the part in red is still healthy. The percentage

of burnt forest can be expressed with one number, whereas a precise binary segmentation map of the fire or burnt areas encoded in *Run Length Encoding* (RLE) would only be a few kiloBytes.

In all these cases and many more, it is not necessary to retrieve original full-resolution images to make informed decisions and take action. Performing image analysis onboard will open up the possibility of more storage and bandwidth-efficient remote sensing. It will also facilitate the development of time-critical applications, and even allow real-time alerts in the future.

1.2 Methods and Challenges

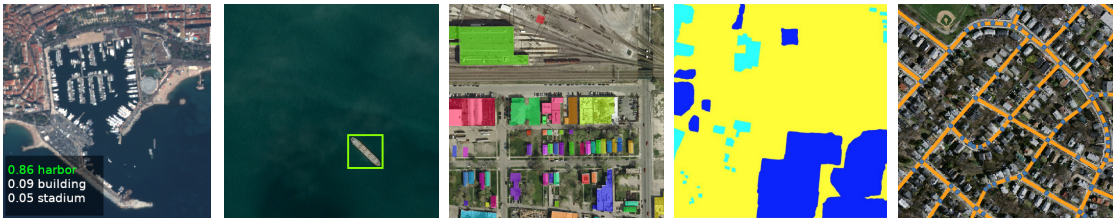
The following sections provide a high-level introduction to Deep Learning and its associated vocabulary, along with a brief history of state-of-the-art methods and their use in remote sensing applications. Each following chapter will then introduce its own specific related works and notions. Finally, we present the objectives of this thesis by exploring the challenges of onboard processing.

1.2.1 Image processing tasks in Remote Sensing

The various image analysis tasks that are performed on remote sensing images and on more general images fall into a few broad categories, depicted in Figure 1.5. They are most commonly described as follows:

- **Image Classification** assigns one or multiple labels to an entire image.
- **Object Detection** locates objects within an image using bounding boxes and classifies them.
- **Instance Segmentation** is similar to object detection, but also assigns a pixel-level mask to each object.
- **Semantic Segmentation** is the task of assigning a label to each pixel of an image, without necessarily assigning it to a specific object instance. This task is typical in Land Use/Land Cover applications. Combining Instance and Semantic segmentation is another task called *Panoptic Segmentation*.
- **Graph Extraction** finds a set of nodes and edges in an image (*e.g.* road graphs). Polygon extraction and polygonal partitioning are also forms of graph extraction.

In this thesis, we focus on semantic segmentation, instance segmentation, and graph extraction. These tasks are particularly challenging in the field of remote sensing, due to the large size and fine details of satellite images.



Classification | Object detection | Instance seg. | Semantic seg. | Graph extraction

Figure 1.5: Various computer vision tasks usually performed on satellite images. Instance seg. image from [Ohl18], semantic seg. image from [SC19].

1.2.2 Deep Learning in Remote Sensing

When remote sensing using satellites was still in its infancy, resolutions were coarse enough that a single pixel belonged to more than a single object. Thus, the spectral signature of each pixel contained a summarized information about broad underlying categories present in the images (*e.g.* "forest", "lake", or "road"). Semantic classification of pixels (*i.e.* semantic segmentation) was performed using traditional computer vision methods based on spectral statistics and texture information [HSD73; SB91; BPS99]. Following works used traditional machine learning techniques such as Support Vector Machines [MIO11], Random Forests [She+20] or Multi-Layer Perceptrons (MLP) [Del+07] for image or pixel-wise classification. In the past decade, as Deep Convolutional Neural Networks (CNNs) became popular within the computer vision community, the remote sensing community also transitioned towards deep learning based methods [Zhu+17].

Convolutional Neural Networks were originally developed for image classification. Most architectures use a similar set of basic functions, called *layers*, that are composed in sequence to create a similar structure, shown in Figure 1.6.

- **Convolutional layers** apply filters corresponding to learned features in a sliding window over the image. The resulting feature maps have high values where the patterns in the filters are matched in the image.
- **Pooling layers** perform an aggregation (often the max function) in a sliding window to create a local summary of the information in feature maps, and reduce the dimension of the image.
- **Linear layers**, also called *Fully Connected* or *Dense* layers perform a learned linear operation, and are used in the final stages of the neural network in order to obtain the final classification.
- **Activation functions**, also called *non-linearities* follow convolutional and linear layers and allow the neural network to approximate arbitrary functions.

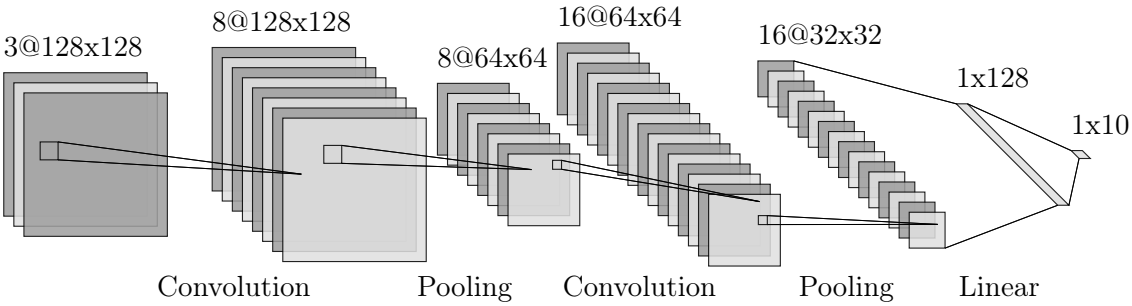


Figure 1.6: Structure of a classical Convolutional Neural Network architecture for image classification.

While the first Convolutional Neural Networks (CNN) were developed in the 90's [LeC+98], they only really gained in popularity when a deep CNN won the ImageNet challenge in 2012 [KSH12], using more than 62 million parameters. In order to train such a neural network, two important requirements needed to be fulfilled. The first one is the availability of large amounts of labeled data to train the neural network, which was provided by ImageNet [Jia+09]. The second one is the availability of powerful *Graphics Processing Units* (GPUs) along with software that allows to run general purpose code on them (CUDA).

Architectures that followed AlexNet [KSH12] featured more and more convolutional layers and parameters, in order to obtain better benchmark scores [SZ14; He+16]. Thus, the computing ressources and quantity of labeled data required to train state-of-the-art CNNs have not diminished. In the case of remote sensing, large training databases have been created for image classification [Lon+21], semantic segmentation [AB20; Mag+17], instance segmentation [GL21], object detection [Lam+18] and graph extraction [VLB18].

1.2.3 On-board image processing

Processing images onboard in order to achieve better data efficiency is actually nothing new. Compression, in particular, has been studied since the beginnings of satellite imagery [RW73]. We refer the reader to surveys for more information on this subject [JTP21; DKS20; Hua11; YVS09]. Nowadays, most satellite systems feature hardware dedicated to image compression, which takes advantage of the high inter-channel correlation in satellite images. However, the semantic information contained in these images is not taken into account.

While the *Deep Learning* techniques presented in the previous section have the ability to extract semantic information, using them on-board is no easy feat. State-of-the-art architectures feature millions, billions or even trillions of parameters. They are typically trained and used on GPUs or TPUs consuming hundreds of

Watts. The power and cooling constraints present in satellites mean that such powerful hardware cannot be used onboard.

The progress made in chip manufacturing over the last decade have not only allowed the creation of GPUs that are orders of magnitude faster than before, but also the creation of devices based on efficient low-power chips with reasonable performance, such as the Nvidia Jetson family of development kits. Now that *Deep Learning* inference is possible on the edge, these low-power accelerators are starting to make their way into satellites (*e.g.* ESA's PhiSat-1). Thus, neural network architectures need to be designed for on-board satellite image processing and specifically tailored towards low-power AI accelerators.

1.3 Contributions and outline

In this section, we describe the contributions of each chapter of this thesis. Note that each chapter addresses a different problem, and thus has its own introduction, related work, methodology and conclusion sections. Nevertheless, Chapters 2 and 3 should be read in order.

Chapter 2 Semantic segmentation methods have made impressive progress with deep learning. However, while achieving higher and higher accuracy, state-of-the-art neural networks overlook the complexity of architectures, which typically feature dozens of millions of trainable parameters. Consequently, these networks require high computational ressources and are mostly not suited to perform on edge devices with tight resource constraints, such as phones, drones, or satellites. In this chapter, we propose two highly-compact neural network architectures for semantic segmentation of images, which are up to 100 000 times less complex than state-of-the-art architectures while approaching their accuracy. To decrease the complexity of existing networks, our main ideas consist in exploiting lightweight encoders and decoders with depth-wise separable convolutions and decreasing memory usage with the removal of skip connections between encoder and decoder. Our architectures are designed to be implemented on a basic FPGA such as the one featured on the Intel Altera Cyclone V family of SoCs. We demonstrate the potential of our solutions in the case of binary segmentation of remote sensing images, in particular for extracting clouds and trees from RGB satellite images. The solutions presented in this chapter have been tested onboard OPS-SAT, a CubeSat launched by ESA in December 2019.

Chapter 3 Traditional Convolutional Neural Networks (CNN) for semantic segmentation of images use 2D convolution operations. While the spatial inductive bias of 2D convolutions allow CNNs to build hierarchical feature representations,

they require that full-size feature maps are used throughout the inference, which incurs high memory usage. This is not ideal for memory and latency-critical applications such as real-time on-board satellite image segmentation. In this chapter, we propose a new neural network architecture for semantic segmentation, "ScannerNet", based on a Recurrent 1D Convolutional architecture. Our network performs a segmentation of the input image line-by-line, and thus reduces the memory footprint and output latency. These characteristics make it ideal for on-the-fly segmentation of images on-board satellites equipped with push broom sensors such as Landsat 8, or satellites with limited computing capabilities, such as CubeSats. We perform cloud segmentation experiments on embedded hardware and show that our method offers a good compromise between accuracy, memory usage and latency.

Chapter 4 While deep learning-based object detection methods traditionally make use of pixel-level masks or bounding boxes, alternative representations such as polygons or active contours have recently emerged. Among these, methods based on the regression of Fourier or Chebyshev coefficients have shown high potential on freeform objects. However, because such methods define object shapes as polar functions, they are limited to star-shaped domains. We address this issue with Smooth Contour Regression (SCR): a method that captures resolution-free object contours as complex periodic functions. The method offers a good compromise between accuracy and compactness thanks to the design of efficient geometric shape priors. We benchmark SCR on the popular COCO 2017 instance segmentation dataset, and show its competitiveness against existing algorithms in the field. In addition, we design a compact version of our network, which we benchmark on embedded hardware with a wide range of power targets, achieving up to real-time performance.

Chapter 5 Automatic road graph extraction from aerial and satellite images is a long-standing challenge. Existing algorithms are either based on pixel-level segmentation followed by vectorization, or on iterative graph construction using next move prediction. Both of these strategies suffer from severe drawbacks, in particular high computing resources and incomplete outputs. By contrast, we propose a method that directly infers the final road graph in a single pass. The key idea consists in combining a Fully Convolutional Network in charge of locating points of interest such as intersections, dead ends and turns, and a Graph Neural Network which predicts links between these points. Such a strategy is more efficient than iterative methods and allows us to streamline the training process by removing the need for generation of starting locations while keeping the training end-to-end. We evaluate our method against existing works on the popular Road-

Tracer dataset and achieve competitive results. We also benchmark the speed of our method and show that it outperforms existing approaches. This opens the possibility of in-flight processing on embedded devices.

Chapter 6 In the last chapter, we conclude this thesis with some final remarks and future directions.

Chapter 2

On-board image segmentation with compact networks

In this chapter, we design efficient Fully Convolutional architectures for on-board semantic segmentation, and apply them to cloud and forest segmentation.

2.1 Introduction

Semantic segmentation of images is a long standing problem in Computer Vision. The popularization of Convolutional Neural Networks (CNN) has led to a lot of progress in this field. Fully Convolutional Networks (FCN) [KFS15] and related architectures, such as the popular U-Net [RFB15], outperformed traditional methods in terms of accuracy while being easy to use.

In the quest towards accuracy, state-of-the-art neural networks often disregard the complexity of their architectures, which typically feature millions of trainable parameters, requiring days of training and gigabytes of hard-drive space. These architectures also require a lot of computing power, such as GPUs consuming hundreds of Watts. Consequently, these architectures cannot be easily used on edge devices such as phones, drones or satellites.

Our goal is to design neural network architectures operating on low-power edge devices with the following objectives:

- **High accuracy.** The architecture should deliver accurate semantic segmentation results.
- **Low complexity.** The architecture should have a low number of parameters to learn.
- **Adaptability.** The architecture should be easily implementable on low-power devices.

Some efficient architectures, such as ESPNet [Meh+18], and convolution modules [How+17] have been proposed for phones and autonomous vehicles. These architectures are however too complex to be exploited on devices with lower power budget and computational resources. For instance, this is the case of FPGA cards embedded into system-on-chips (SoCs) on satellites, which have limited floating-point capabilities and are restricted by their amount of hardware logic.

In this chapter, we present two highly-compact neural network architectures for semantic segmentation, which are up to 100 000 times less complex than state-of-the-art architectures while approaching their accuracy. We propose two fully-convolutional neural networks, C-FCN and C-UNet, which are compact versions of the popular FCN [KFS15] and U-Net [RFB15] architectures. To decrease the complexity of existing networks, our main ideas consist in exploiting lightweight encoders and decoders with depth-wise separable convolutions and decreasing memory usage with the removal of skip connections between encoder and decoder. Our architectures are designed to be implemented on a basic FPGA such as the one featured on the Intel Altera Cyclone V family of SoCs. We demonstrate the potential of our solutions in the case of binary segmentation of remote sensing images, in particular for extracting clouds and trees from RGB satellite images. Our methods reach 95% accuracy in cloud segmentation on 38-Cloud [MKS18] and CloudPeru2 [MHT18] datasets, and 83% in forest segmentation on the EOLearn Slovenia 2017 dataset.

2.2 Related works

We first review prior work in two related aspects of our goal: neural networks for semantic segmentation and neural network inference on edge devices.

2.2.1 Neural networks for semantic segmentation

Many deep learning methods have been proposed to improve performance of segmentation networks, in terms of accuracy on popular benchmarks [Eve+10; Lin+14; Mot+14] and speed, with the goal of real-time semantic segmentation [Sia+18]. These methods are compared in recent surveys [Guo+18; Gar+18; Sia+18] without studying the networks from a complexity point of view.

Encoder-decoders

Most of the popular segmentation neural networks, such as U-Net [RFB15], SegNet [BKC17], or DeepLabv3+ [Che+18] adopt an encoder-decoder architecture.

The encoder part of the network, also called backbone, is typically a Deep Convolutional Neural Network composed of multiple stages of convolution operations, separated by pooling operations. This allows the encoder to capture high-level features at different scales. It is common to use an existing CNN architecture such as ResNet-101 [He+16], VGG16 [SZ14], or MobileNet [How+17] as a backbone, since pre-trained weights on databases such as ImageNet [Jia+09] are available for transfer learning. However, while achieving high accuracy, these encoders are complex and not suited for inference on edge devices. Decoders are in charge of upsampling the result to get an output that has the same size as the original image. Deconvolutions, also called transposed convolutions or up-convolutions, introduced by H. Noh *et al.* [NHH15] in the context of semantic segmentation, are used to learn how images should be upscaled.

Skip connections

Skip connections are often used between convolution blocks, i.e. short skips, in architectures such as residual networks [He+16], and/or between the encoder and the decoder, i.e. long skips, in architectures such as U-Net [RFB15]. These connections are performed by concatenating feature maps from different parts of the network in order to retain some information. In the case of segmentation networks, skip connections are used to keep high-frequency information (e.g. corners of buildings, borders of objects) to obtain a more precise upscaling by the decoder. While skip connections can yield good results in practice, they are often memory consuming, especially the long skips, as they require feature maps to be kept in memory for a long time. Consequently, they cannot easily be used on edge devices with limited memory. In particular, we show in Section 2.4.6 that they are not necessarily useful in all use cases.

2.2.2 Neural networks on edge devices

Efficient convolution modules

Several low-complexity convolution modules have been developed, with the goal of reducing power consumption and increasing inference speed on low-end or embedded devices, such as phones. This is the case of Inception [Sze+15], Xception [Cho17], ShuffleNet [Zha+18], or ESP [Meh+18]. These modules are based on the principle of convolution factorization and turn classical convolution operations into multiple simpler convolutions. S. Mehta *et al.* provide a detailed analysis of these modules in [Meh+18]. However, we cannot use them as they still feature too many trainable parameters and often require to store the results of multiple convolution operations performed in parallel or short skip connections. In contrast, we use

Depth-wise Separable Convolutions, introduced in [Sif14] and used in MobileNets [How+17].

Neural Network simplification

Several techniques have been developed to simplify neural networks for systems with tight resource constraints. Training and inference using fixed-point or integer arithmetic and quantization is an active research topic [LTA16; Jac+18; Wu+18], since floating-point operations are costly on many embedded systems such as FPGAs, and trade-offs have to be made between accuracy and speed. Neural network pruning, on the other hand, is the process of removing some weights [LDS90] or entire convolution filters [Li+16; Mol+16] and their associated feature maps in a neural network, in order to extract a functional "sub-network" that has a lower computational complexity and similar accuracy. Some recent works focus on resource constrained Neural Architecture Search [XMS19]. A lot of work has been done on enabling and accelerating NN inference on FPGA [Abd+18], with tools being released to automatically generate HDL code for any neural network architecture [Ham18], with automated use of quantization and other simplification techniques. These approaches are orthogonal to our work as they could be applied to any neural network.

2.3 Proposed architectures

We now detail our compact neural networks, C-UNet and C-FCN. These architectures are Fully Convolutional. Thus the number of parameters of the networks does not depend on the size of the input images. In particular, the training steps and the testing steps can be performed on different image sizes.

2.3.1 C-UNet

As illustrated in Figure 2.1, C-UNet is a compact version of the popular U-Net architecture [RFB15].

To create a shallower network, three convolution stages have been removed with respect to the original architecture. Removing stages narrows the field-of-view of the network, but also strongly reduces the computational cost as the number of feature maps is typically multiplied by a factor 2 at each stage. For instance, the original U-Net [RFB15] has 1024 feature maps at the last encoder stage. A standard 3x3 convolution at this stage uses $3 \times 3 \times 1024 \times 1024 + 1024 = 9438208$ parameters. The number of stages to remove has been chosen after empirical experimental validation, 3 being a good compromise between field-of-view and

number of parameters. Indeed, we found experimentally that we could train a neural network to classify objects such as clouds and trees in 28x28 pixel images with good accuracy. The encoder in C-UNet has a receptive field of 50x50 pixels and is consequently more than capable of reaching the same accuracy.

Some of the standard convolution layers have also been replaced by depth-wise separable convolutions. Introduced in [How+17], they allow a classical convolution to be split into a depth-wise convolution and a point-wise convolution, to use the fact that inter-channel features are often not related to spatial features. Since the convolution kernel is the same for all input feature maps in a depth-wise convolution, this allows us to reduce the number of learned parameters significantly. For example, a standard 3x3 convolution with 16 input and output feature maps has 2320 trainable weights, while a depth-wise separable convolution of the same size only has 416 trainable weights.

The number of filters per convolution has been chosen so that C-UNet has 51 113 parameters, *i.e.* around 500 times less than the architecture implemented in [MKS18]. C-UNet then has 8 filters per convolution at the first stage and this number is doubled at every stage in the encoder, and divided by two at every stage in the decoder, as in the original U-Net architecture [RFB15].

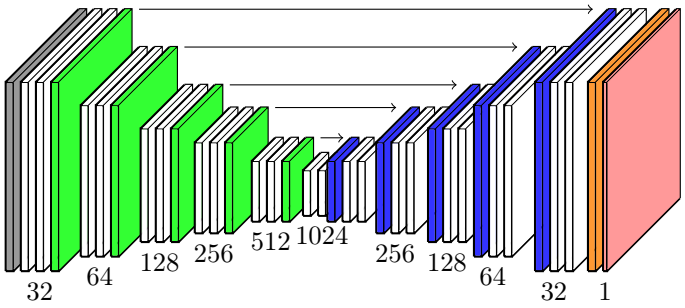
We also propose a variant of C-UNet, called C-UNet++, with an even more compact architecture. C-UNet++ contains 9 129 parameters, *i.e.* 3000 times less parameters than the architecture implemented in [MKS18]. As illustrated in Figure 2.1, the number of convolution layers has been reduced to one per stage, and only the first stage has standard convolutions. This gives C-UNet++ a receptive field of 22x22 pixels.

Note that skip connections between corresponding stages in encoder and decoder parts could be considered. However, we show in Section 2.4.6 that they are not necessarily useful, as adding them may not significantly increase performance and may require a lot of memory depending on the input image size.

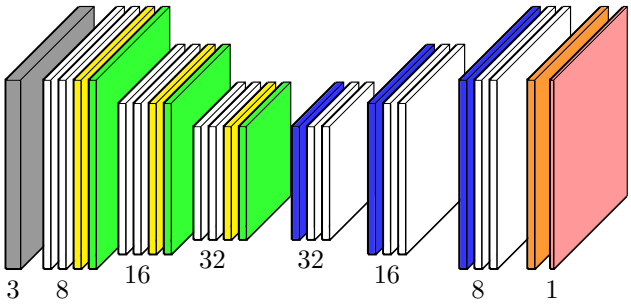
2.3.2 C-FCN

The second proposed architecture, called C-FCN, relies upon a lightweight CNN encoder and a bilinear upsampler, as illustrated in Figure 2.2. The encoder has 3 stages, with a single convolution per stage. A 1x1 convolution then generates a class heatmap which is finally upsampled by the bilinear upsampler. C-FCN uses depthwise separable convolutions to maximize the number of convolution filters we can use within our parameter budget, which is of 1 438 parameters for this architecture.

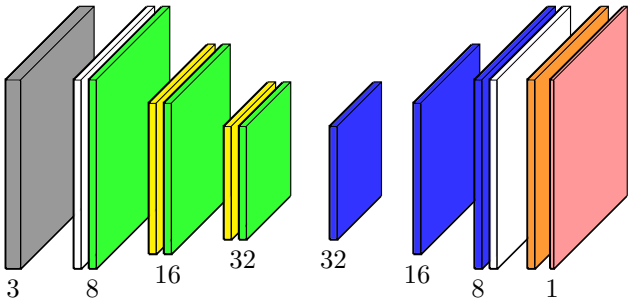
We also propose a variant of C-FCN with the same depth, called C-FCN++, which is designed to be the smallest viable architecture with only 273 parameters.



(a) U-Net



(b) C-UNet



(c) C-UNet++

Figure 2.1: U-Net [RFB15] compared to C-UNet and C-UNet++. Gray: input, white: 3x3 convolution with ReLU activation, yellow: depthwise separable 3x3 convolution with ReLU activation, green: 2x2 max pooling, orange: 1x1 convolution with sigmoid activation, blue: 2x2 deconvolution, red: output, arrows: skip connections. Numbers indicate the number of feature maps at each stage.

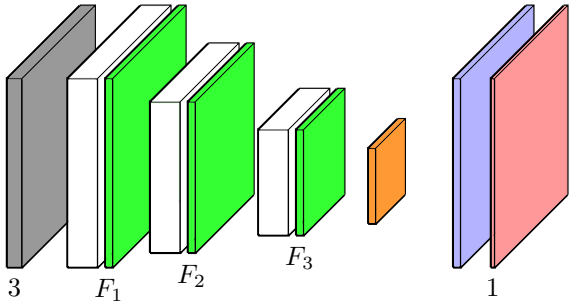


Figure 2.2: C-FCN and C-FCN++ architectures. Gray: input, white: conv3x3 ReLU, green: 2x2 max pool, orange: conv1x1 sigmoid, blue: 4x4 bilinear upscaling.

Architecture	F_1	F_2	F_3	Depthwise	Atrous	N_{param}
C-FCN	10	20	40	Yes	No	1438
C-FCN++	5	2	2	No	1^{st}conv.	273

Table 2.1: Characteristics of C-FCN and C-FCN++. Depthwise = usage of depthwise separable convolutions. F_x is the number of convolution filters and feature maps at stage x.

The first layer is an atrous convolution layer with a dilation rate of 2, which broadens the receptive field of the network a little bit [YK15].

The number of filters of C-FCN and its variant C-FCN++ are showed in Table 2.1.

2.4 Experiments on cloud segmentation

In this section, we evaluate the performance of our architectures for cloud extraction in RGB remote sensing images.

Change detection and related topics, *e.g.* anomaly detection, have always been of interest for the remote sensing community, as they are used in practical domains, such as natural disaster protection, urban development analysis, deforestation monitoring, and so on. About 52% of the Earth's surface is covered by clouds [Dow05]. While it is useful in some fields such as meteorology, detecting cloud movements and changes is irrelevant in many others, where images containing clouds and their shadows are simply useless. Thus, detecting clouds and removing them is a first step towards detecting relevant changes.

Accurate detection of clouds is not an easy task, especially when a limited number of spectral bands is available, as clouds share the same radiometric properties

as snow, for example. Currently, commonly-used cloud segmentation methods are either threshold-based or handcrafted, with the most popular ones being Fmask [ZWW15] and Haze Optimized Transform [ZGC02]. These cannot be used in the case of OPS-SAT as its sensor only features RGB channels, which means we do not have the required spectral bands at our disposal. These methods are pixel-based (each pixel is independant for cloud detection), whereas we need to use the context of adjacent pixels (e.g. presence of a cloud shadow to differenciate cloud from snow), which CNN are able to do, thanks to convolution and pooling operations. Performing real-time cloud segmentation directly onboard and removing useless cloud-covered images can allow significant bandwidth, storage and computation time savings on the ground.

M. Hughes *et al.* [Hug+14] were, to our knowledge, the first to use a neural network-based approach for cloud segmentation in Landsat-8 images. However, convolutional neural networks were not used. A lot of recent work has been done on cloud segmentation using CNN [MKS18; MS19; MHT18; Drö+18; Liu+19]. However, all these methods use compute-heavy architectures on hyperspectral images and are designed to be used with powerful hardware on the ground, and thus, are not compatible with our use-case. Cloud segmentation using neural networks has already been integrated in an ARTSU CubeSat mission by Z. Zhang *et al.* [ZXS18]. S. Ghassemi *et al.* [Gha+19] have proposed a small strided U-Net architecture for onboard cloud segmentation. Their architectures are still too big for our use cases and are designed to work on 4-band images (RGB + NIR). Nevertheless, we include a 3-band implementation of MobUNet [ZXS18], MobDeconvNet [ZXS18] and "Plain+" strided U-Net [Gha+19] architectures in our comparison.

2.4.1 Datasets

We use publicly available cloud segmentation datasets to train and test the models.

We use the Cloud-38 dataset, introduced by S. Mohajerani *et al.* [MS19]. The dataset is composed of 4-band Landsat-8 images with a 30m resolution. We only use RGB bands. Since some of the training scenes have black borders in the original dataset, we remove all the patches that have more than 50% of black pixels. The resulting training and validation set is composed of 4 821 patches of size 384x384. The testing set is composed of 9 200 patches.

We also use the CloudPeru2 dataset, introduced by G. Morales *et al.* [MHT18]. This dataset is composed of 4-band PERUSAT-1 images with a 3m resolution. It features 22 400 images of size 512x512. We use 10% of these images for testing. Test images have been chosen randomly and are the same for all of our tests.

2.4.2 Training procedure

All models were implemented using the Keras framework using Tensorflow 1.13 as a backend in Python 3.6.

Models are trained using a round-based scheme. Training sets are randomly shuffled and split into training (85%), and validation (15%) sets at the beginning of each round. Within a round, all models are then trained on the same training and validation sets for a maximum of 150 epochs. At the end of all rounds, the best network for each architecture is selected using the testing set. This was not ideal in hindsight and the best netword should have been selected using the validation set. However, since we apply the same treatment to all methods, the results are still comparable with each other.

Data is augmented using random horizontal and vertical flips, as well as random rotations of 5 degrees maximum. The same random seed is used for all architectures. Models were trained for 4 rounds with batch-size 8 and the Adam optimizer with a learning rate of 0.001. Binary crossentropy loss is used for all networks. Early stopping is used with a patience of 15 and a maximum number of epochs of 150.

This training scheme takes around 10h per architecture, per round, per dataset on an Nvidia GTX 1080 Ti graphics card, which brings our total training time to around 1160h. Training could have been faster, as the card's VRAM was barely being used when training really small models, but we wanted to use the same training parameters (batch-size) for every model, to get the fairest possible comparison. The largest batch-size we could use to train the biggest network (U-Net) is 8.

2.4.3 Comparative architectures

We compare our networks to different semantic segmentation architectures from the litterature, that we re-implemented:

- U-Net for cloud segmentation, as presented by S. Mohajerani *et al.* [MKS18], which we use as a "baseline" for comparison, since U-Net, originally presented by Ronneberger *et al.* [RFB15] for the segmentation of medical images is a well-known and easy to implement architecture that has proved useful for many segmentation tasks,
- ESPNet_A, introduced by S. Mehta *et al.* [Meh+18], which is a state-of-the-art segmentation network, in terms of efficiency and accuracy. We use $K = 5, \alpha_2 = 2, \alpha_3 = 5$.
- MobUNet and MobDeconvNet, introduced by Z. Zhang *et al.* [ZXS18], are small versions of U-Net [RFB15] and Deconv-Net [NHH15] for onboard cloud segmentation, that use depth-wise separable convolutions [How+17],

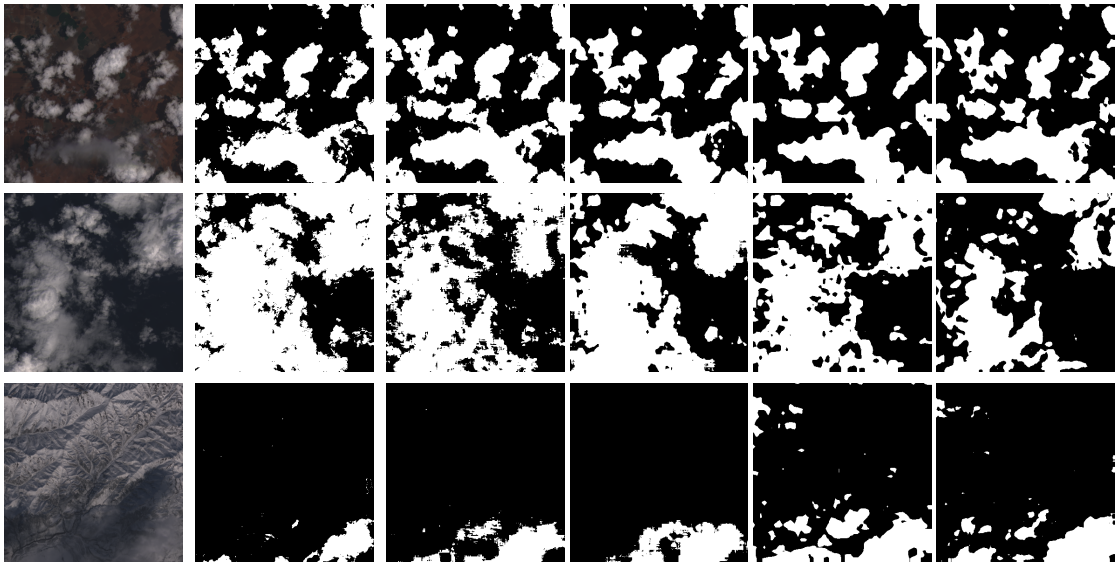


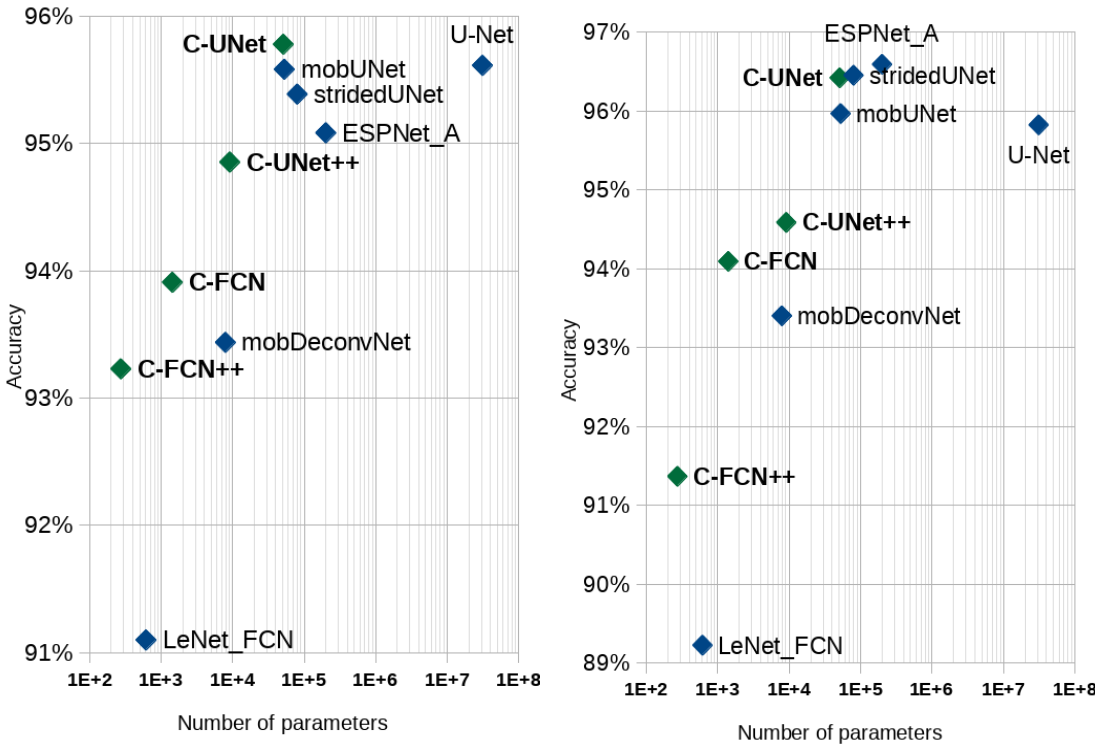
Figure 2.3: Qualitative results of C-UNet, C-FCN, and C-FCN++ on 38-Cloud dataset, compared to Ground Truth (GT) and U-Net [MKS18]. Our networks in bold font. Our networks give good results and are not fooled by snow on the third patch. C-FCN and C-FCN++ results appear very smooth compared to others because the segmentation map output by the network is 4 times smaller than U-Net's and upsampled with a bilinear upsampling. Thus, C-UNet produces a more refined segmentation than C-FCN.

- StridedUNet, introduced by Ghassemi *et al.* [Gha+19] for onboard cloud segmentation, which uses strided convolutions for downsampling,
- LeNetFCN, an FCN variation of the well-known LeNet-5 architecture [LeC+98], which was originally created for written number classification. We replace the final Fully-Connected (dense) layers by a 1x1 2D Convolution with a sigmoid activation, in order to output a segmentation map. LeNetFCN architecture looks like the "C-FCN" architecture (see Figure 2.2) with $F_1 = 3, F_2 = 5$, except it uses 5x5 convolutions instead of 3x3 convolutions.

We evaluate models using standard segmentation metrics: Total Accuracy, Precision, Recall, Specificity and Jaccard Index (IoU), as defined in [MS19].

2.4.4 Qualitative results.

Figure 2.3 shows inference results of our networks on three image patches taken from the 38-Cloud test set, compared to the ground truth and U-Net. Our networks provide a visually good result, even on difficult terrains such as snow. The



(a) 38-Cloud (b) CloudPeru2

Figure 2.4: Test accuracy with respect to number of parameters on the two tested datasets. Our networks are in green and bold font, others in blue. Number of parameters is on a log scale. Our networks offer good accuracy at varying levels of complexity.

segmentation masks produced appear smooth, especially for C-FCN/C-FCN++ because of the bilinear upsampling. This is not an issue to estimate cloud coverage as a percentage.

2.4.5 Quantitative results.

Table 2.2 shows accuracy metrics on 38-Cloud and CloudPeru2. We include the Fmask accuracy results from S. Mohajerani *et al.* [MS19] for 38-Cloud, since our test dataset and metrics are the same.

The best network under 1500 parameters is C-FCN. In particular, this network matches Fmask's accuracy (within 1%), and achieve much better precision and IoU. This network also exceeds the performance of our implementation of mobDeconvNet [ZXS18], using 4x less parameters. Our C-FCN++ matches the

Model	Acc.	Prec.	Rec.	Spec.	Jacc.
LeNet_FCN	91.10	86.00	84.34	94.03	74.16
C-FCN++	93.23	91.67	85.45	96.62	79.30
mobDeconvNet [ZXS18]	93.44	92.04	85.75	96.78	79.83
C-FCN	93.91	91.23	88.39	96.31	81.47
C-UNet++	94.85	94.18	88.48	97.63	83.89
ESPNet_A [Meh+18]	95.08	93.94	89.55	97.49	84.66
stridedUNet [Gha+19]	95.39	94.21	90.36	97.58	85.60
mobUNet [ZXS18]	95.58	95.37	89.78	98.11	86.03
U-Net [MKS18]	95.61	95.69	89.56	98.25	86.08
C-UNet	**95.78**	**96.53**	89.27	**98.61**	**86.50**
Fmask [ZWW15]	94.89	77.71	**97.22**	93.96	75.16
LeNet_FCN	89.23	93.11	83.27	94.52	78.44
C-FCN++	91.37	93.50	87.76	94.58	82.71
mobDeconvNet [ZXS18]	93.40	94.98	90.78	95.73	86.63
C-FCN	94.09	95.15	92.15	95.82	88.01
C-UNet++	94.59	96.03	92.31	96.61	88.92
U-Net [MKS18]	95.82	96.29	94.78	96.75	91.44
mobUNet [ZXS18]	95.97	**97.20**	94.14	**97.59**	91.66
C-UNet	96.42	96.54	95.83	96.94	92.65
stridedUNet [Gha+19]	96.45	96.49	95.95	96.89	92.72
ESPNet_A [Meh+18]	**96.59**	96.45	**96.30**	96.85	**93.01**

Table 2.2: 38-Cloud (top) and CloudPeru2 (bottom) results. Our networks in bold font.

performance of mobDeconvNet while using only 273 parameters. Our C-UNet matches the performance of our implementation of mobUNet [ZXS18], while using less memory thanks to the lack of skip connections. It also exceeds the performance of our implementation of StridedUNet [Gha+19], while being 20% smaller in terms of number of parameters. We see similar results on CloudPeru2 as on 38-Cloud. Our networks provide good results on this high-resolution dataset (3m). Figure 2.4 shows the relation between the number of parameters of a network and its accuracy. Our networks match or exceed the performance or other methods at each complexity level. There is a point of diminishing return in this task: our C-UNet performs about the same as a 500 times bigger U-Net.

2.4.6 Impact of skip connections.

We tried adding skip connections in C-UNet and C-UNet++, but did not find a significant difference in performance metrics in the case of cloud segmentation, as shown in Table 2.3. Skip connections are useful to keep high frequency information and use it in the decoder part of the network, since the encoder part tends to act like a low-pass filter because of successive convolutions. Clouds do not have much high frequency information, which is maybe why skip connections were not particularly useful in our case. This means that a significant amount of memory can be saved during network inference by removing skip connections in use cases where they are not useful, as shown in Table 2.4. This can allow bigger networks to be used on systems with tight resource constraints.

C-UNet	Acc.	Prec.	Rec.	Spec.	Jac.
with skips	**95.90**	96.22	**90.01**	98.46	**86.93**
w/o skips	95.78	**96.53**	89.27	**98.61**	86.50
C-UNet++	Acc.	Prec.	Rec.	Spec.	Jac.
with skips	94.51	92.30	**89.33**	96.76	83.14
w/o skips	**94.85**	**94.18**	88.48	**97.63**	**83.89**

Table 2.3: Impact of skip connections on performance of C-UNet(++) on 38-Cloud dataset. We do not see a significant difference in accuracy when using skip connections for this cloud segmentation task.

Architecture / Image size	384x384	2000x2000
U-Net [MKS18]	34.9 MB	236.5 MB
StridedUNet [Gha+19]	4.22 MB	114.4 MB
MobUNet [ZXS18]	1.69 MB	45.8 MB
C-UNet with skips	1.97 MB	53.4 MB
C-UNet++ with skips	1.97 MB	53.4 MB
C-UNet without skips	**0 MB**	**0 MB**
C-UNet++ without skips	**0 MB**	**0 MB**

Table 2.4: Memory footprint of skip connections for 32-bit float inference (in Megabytes), computed for two images sizes.

2.4.7 Performance on a low-power processor

Many edge devices use ARM-based SoCs, which are known for their efficiency. This is why we chose the Raspberry Pi, an ARM computer the size of a credit

Architecture	N_{params}	FLOPs	Storage (MB)
C-FCN++	273	4 654	0.047
LeNet_FCN	614	10 455	0.049
C-FCN	1 438	24 233	0.066
mobDeconvNet [ZXS18]	7 919	134 256	0.149
C-UNet++	9 129	154 800	0.172
C-UNet	51 113	867 712	0.735
mobUNet [ZXS18]	52 657	893 882	0.724
StridedUNet [Gha+19]	79 785	1 355 704	1.0
ESPNet_A [Meh+18]	200 193	3 399 310	2.7
U-Net [MKS18]	31 094 497	528 578 588	357

Table 2.5: Network parameters, FLOPs (FLoating-point OPerationS) and storage size. Our architectures in bold font. FLOPs computed for a 384x384 input using Tensorflow's profiler. We see that the number of FLOPs is correlated to the number of paramters. Storage size is the size of the Keras h5 file for each model in MegaBytes.

card, for our testing. Our particular board is the Raspberry Pi 4 model B, which features a Quad core Cortex-A72 CPU clocked at 1.5GHz and 2GB of LPDDR4 SDRAM. We feel that this board is representative of the kind of power a typical low-power system could have, as this board consumes 3W when idle and 6W under load on average. We use Tensorflow 1.13.1 with Python 3.7.3 on Raspbian 10.0 to run our evaluation code on the board.

The number of parameters, or trainable weights, of an architecture is an important metric, as it is not only linked to the storage space needed for these parameters but also (albeit loosely) correlated to computation time since each weight must be used in computation at some point. On systems such as FPGAs where networks can be "hardcoded" into a chip, the number of parameters is also linked to the amount of hardware logic that will be used by a network.

Table 2.5 shows the size used by the tested architectures in terms of number of parameters and storage space, as well as the number of FLOPs (Floating Point Operations) used for one forward pass on a 384x384 image. We can see that the number of FLOPs is indeed correlated to the number of parameters.

Table 2.6 shows the execution time results we obtained. We can see that our networks do not run out of memory when trying to process relatively big images of 2048x2048 pixels at once. Even with a Python 32-bit float implementation, our architectures allow fast processing of images. This processing could be even faster by using Tensorflow Lite and quantized weights.

We notice a strange behaviour with stridedUNet, which performs significantly

Model	384x384	1024x1024	2048x2048
C-FCN++	**0.130**	0.773	**2.663**
stridedUNet [Gha+19]	0.133	**0.752**	2.666
C-FCN	0.148	0.903	3.017
C-UNet++	0.204	1.242	4.295
C-UNet	0.404	2.867	10.355
mobUNet [ZXS18]	0.688	5.507	OOM
mobDeconvNet [ZXS18]	0.755	N/A	N/A
ESPNet_A [Meh+18]	1.878	15.940	OOM
U-Net [MKS18]	5.035	OOM	OOM

Table 2.6: Execution time on images of different sizes in seconds on Raspberry Pi 4, averaged over 20 iterations. OOM = Out of Memory, N/A = mobDeconvNet is not an FCN and thus cannot be executed on images of different sizes than training images.

faster than networks that require less FLOPs. We suspect that Tensorflow, or at least the ARM version of it, was lacking some optimizations in the case of depthwise separable and atrous convolutions.

2.4.8 Quantization

In order to analyze the effects of naive 16-bit and 8-bit quantization of a neural network's weights, we apply the same steps as the ones used for fixed-point conversion. First we scale the weights by 2^8 for 16-bit or 2^4 for 8-bit. Then, we convert the weights to 16-bit or 8-bit signed integers. Finally, we reverse the scaling and convert the weights back to 32-bit float for evaluation.

Figure 2.5 shows the impact of quantization on the behavior of our networks. We used the previously selected networks, *i.e.* best performing out of all rounds, for this test. We see that 16-bit quantization has a minimal impact of 1.69% on average, and can even bring a small boost in accuracy, although we do not have an explanation for this phenomenon. 8-bit quantization has a much bigger impact on performance of 10.79% on average. It should be noted that some networks simply "stop working" when quantized in this manner. Our FCN-based networks do not appear to suffer extensively from 8-bit quantization.

2.4.9 Adaptability to FPGA

We successfully implemented our LeNet-based FCN, on an Altera Cyclone V 5CSXC6 FPGA with a dataflow/streaming architecture and fixed-point quanti-

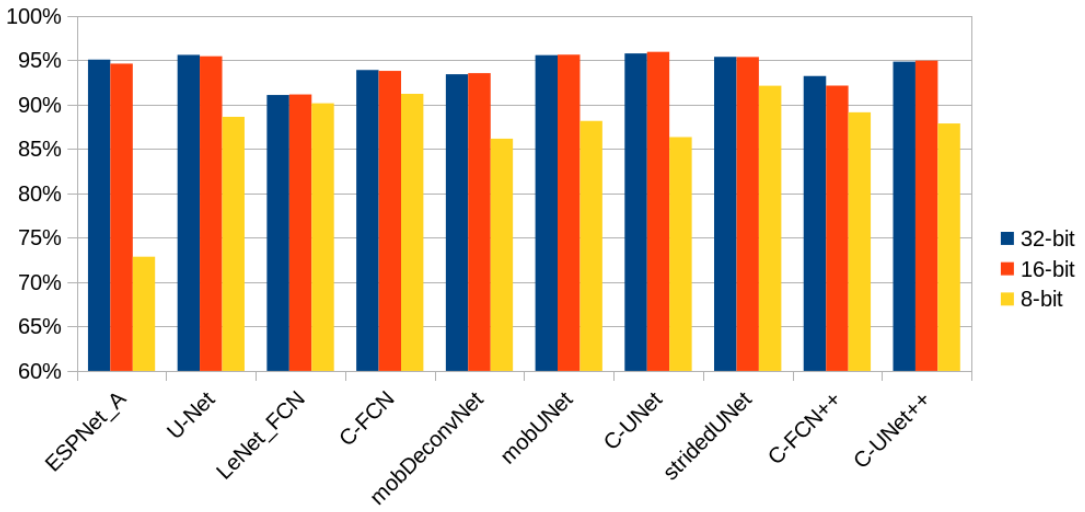


Figure 2.5: Test accuracy of quantized models on the 38-Cloud dataset.

zation tool based on VGT [Ham18], which automatically generates HDL code for a given neural network architecture. This FPGA is included onboard OPS-SAT, a CubeSat satellite that was launched in 2019 by the European Space Agency. The satellite features an RGB camera with a 53m ground sampling distance.

We used a network pre-trained on 38-Cloud and CloudPeru2 images, and fine-tuned it on images captured by OPS-SAT, which were annotated manually. A selection of images is shown on Figure 2.6. The images look very different than the ones included in the two datasets, hence the need for fine-tuning. The network uses 92% of the Adaptive Logic Modules (ALMs), which are the basic FPGA building blocks. The processing of a 28x28 pixel image is done in 25µs and we reach an F-score of 0.72 with a 19-bit quantization. The power consumption measured in orbit was 1.8W. Qualitative inference results computed in-flight in early 2021 are shown in Figure 2.7.

2.5 Experiments on forest segmentation

To provide a comparison on another use case, we also evaluate our models on a forest segmentation task, which is also important in remote sensing, for applications such as deforestation monitoring. Although vegetation detection is usually done using dedicated spectral bands on earth observation satellites, it would be useful to be able to perform it on devices that do not usually possess these bands, such as drones or nanosats (*e.g.* OPS-SAT). Neural networks have already been used suc-

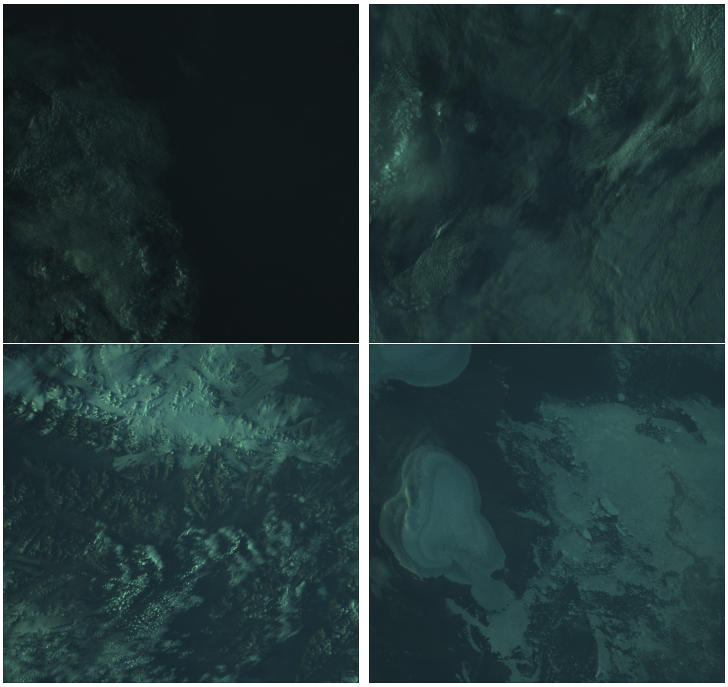
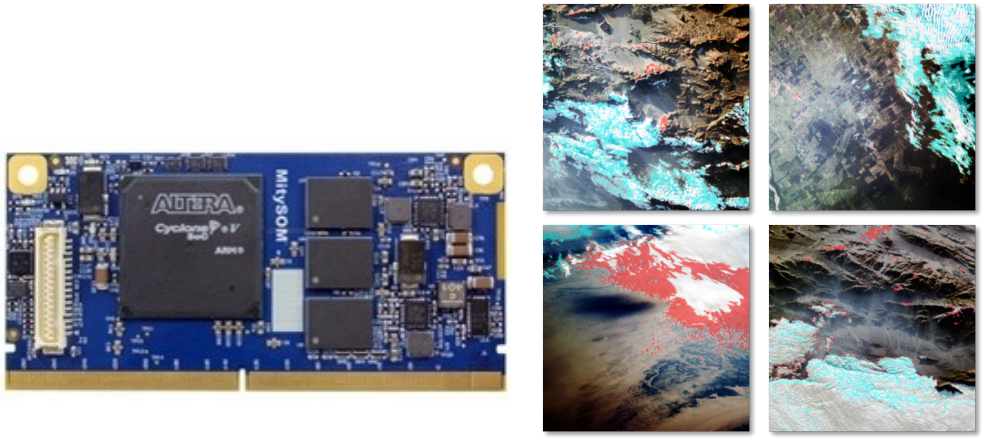


Figure 2.6: Images captured by OPS-SAT and used for fine-tuning of the FCN.



(a) Altera FPGA featured on OPS-SAT (b) Inference done onboard in 2021

Figure 2.7: We performed in-flight cloud segmentation with an FCN using the FPGA of OPS-SAT. Red pixels: false positives. Blue pixels: false negatives. Other pixels: correct segmentation.

Model	Acc.	Prec.	Rec.	Spec.	Jacc.
LeNetFCN	77.10	67.93	63.57	84.22	48.89
C-FCN++	77.40	69.52	61.28	85.87	48.31
C-FCN	78.87	68.95	70.38	83.33	53.44
mobDeconvNet [ZXS18]	80.61	71.52	72.68	84.78	56.36
C-UNet++	80.62	72.92	69.61	86.41	55.31
stridedUNet [Gha+19]	80.67	72.52	70.70	85.92	55.77
C-UNet	83.33	76.79	73.99	**88.24**	60.47
U-Net [MKS18]	84.04	73.37	**84.28**	83.92	**64.54**
mobUNet [ZXS18]	**84.27**	**75.19**	81.13	85.92	63.99

Table 2.7: Slovenia 2017 forest segmentation results (10m resolution).

cessfully for land cover and vegetation segmentation [Zhu+17; KP18]. However, to our knowledge, these tasks have never been done directly onboard.

For this experiment, we use the EOLearn Slovenia 2017 dataset [Sin], which is composed of multiple Sentinel acquisitions of Slovenia over 2017, with pixel-wise land cover ground truth for 10 classes, as well as cloud masks. We only use the forest class and the RGB bands. This dataset is originally split into 293 1000x1000 patches. We remove images that have more than 10% of clouds and split the remaining images into 500x500 pixel tiles. We use 24 patches as a test set. This means our training/validation set has 12 629 tiles, and our test set has 3 320 tiles. We use the same training scheme and settings as in our cloud segmentation experiments, which we detailed in 2.4.2.

In this test, the performance of our networks, as shown in Table 2.7, is comparable to the other networks, which proves their adaptability to different tasks. In particular, C-UNet matches the performance of U-Net [MKS18], and C-UNet++ matches the performance of mobDeconvNet[ZXS18].

2.6 Conclusion

In this chapter, we introduced lightweight neural network architectures for on-board semantic segmentation, and compared them to state-of-the-art architectures in the context of 3-band cloud and forest segmentation. We showed that our networks match the performance of Fmask, a popular cloud segmentation method, while only using 3 bands and at a low computational cost that allow them to be implemented on systems with restrictive power and storage constraints. We also match the performance of network architectures that have previously been used for on-board cloud detection, while using less memory and FLOPs. We talked about the usefulness of depth-wise separable convolutions for implementation of neural

networks on edge devices and discussed the memory footprint of skip connections.

In addition, we explored the performance loss introduced by naive quantization of neural networks for fixed-point inference. We performed experiments on a low-power ARM-based SoC and the FPGA of a satellite. In 2021, our team was the first to remotely upload neural network weights to the FPGA of a satellite in orbit.

Our work has some limitations which leave room for future work. We only tested our architectures on binary image segmentation, and while this is enough for our test cases, many applications require multi-class image segmentation.

Chapter 3

Recurrent convolutional networks for semantic segmentation

In this chapter, we propose a recurrent convolutional network for onboard semantic segmentation in satellites equipped with push broom sensors. This chapter builds on the work from Chapter 2.

3.1 Introduction

Semantic segmentation is not only crucial in remote sensing applications such as cloud segmentation [Bah+19; Fér+21], land use/land cover estimation [Xu+19b], or road mapping [BBL21], but also in other domains, such as medical imagery [RFB15] and self-driving [Cor+16]. In this task, the output segmentation maps have the same resolution as the input images. This makes semantic segmentation a generally more demanding task in terms of computation than image classification.

As a consequence of the increasing demand for edge processing, techniques such as distillation [HVD15] and quantization [Hub+17] have been used as a way to port deep learning models onto low-power devices like UAVs, smartphones or satellites. However, semantic segmentation models remain very compute intensive, and dedicated architectures with a low number of convolution filters had to be developed in conjunction with these techniques in order to be used on low-power satellites, as shown in Chapter 2. Moreover, many earth observation satellites feature push broom sensors. These sensors acquire images of varying height line-by-line (i.e. by "scanning" the ground). In cases where the acquisition runs for a long time, the images created in this way can become too large to be efficiently processed in memory by traditional 2D convolutional neural networks. Finally, since CNNs traditionally process the whole image at once, there is a significant latency between the acquisition and the output of the result, since we have to wait

until the whole image is acquired and processed before viewing the result.

Traditional Convolutional Neural Networks (CNN) for semantic segmentation of images use 2D convolution operations. While the spatial inductive bias of 2D convolutions allow CNNs to build hierarchical feature representations, they require that full-size feature maps are used throughout the inference, which incurs high memory usage. This is not ideal for memory and latency-critical applications such as real-time on-board satellite image segmentation.

In this chapter, we propose a new neural network architecture for semantic segmentation, "ScannerNet", based on a one-dimensional Convolutional LSTM [HS97; Xin+15] (ConvLSTM). Our network parses the input image line-by-line, and outputs the result at the same time. Thus, it only keeps the necessary information in memory and achieves a very low latency. These characteristics make it ideal for on-the-fly segmentation of images on-board satellites equipped with push broom sensors such as Landsat-8, or satellites with limited computing capabilities, such as Cubesats. We perform cloud segmentation experiments on embedded hardware and show that our method offers a good compromise between accuracy, memory usage and latency.

3.2 Method

In this section, we provide a description of our architecture, shown on Figure 3.1. Our architecture is based on a two-layer 1D Convolutional LSTM (ConvLSTM), inspired by the work of Shi et al. [Xin+15] on 2D ConvLSTM. We implement the ConvLSTM cells without self-loops as in [HS97]:

$$\begin{aligned} i_t &= \sigma(\text{Conv1D}([x_t; h_{t-1}], W_i) + b_i) \\ f_t &= \sigma(\text{Conv1D}([x_t; h_{t-1}], W_f) + b_f) \\ o_t &= \sigma(\text{Conv1D}([x_t; h_{t-1}], W_o) + b_o) \\ g_t &= \tanh(\text{Conv1D}([x_t; h_{t-1}], W_g) + b_g) \\ c_t &= f_t \circ c_{t-1} + i_t \circ g_t \\ h_t &= \tanh(c_t) \circ o_t \end{aligned} \tag{3.1}$$

where $\circ$ is the Hadamard product, σ is the sigmoid function, $[x_t; h_{t-1}]$ is the concatenation of the input line x_t at step t and previous hidden state of the cell h_{t-1}. We notice that all the convolution operations in Equation 3.1 can be optimized by combining them into a single 1D convolution with weights $W = [W_i; W_f; W_o; W_g]$, $b = [b_i; b_f; b_o; b_g]$ and output $y = [i_t; f_t; o_t; g_t]$. We denote as F_1 and F_2 the number of feature maps of the hidden states in layer 1 and 2, respectively. The final output line y_t is generated by a 1x1 convolution that maps the F_2 feature maps of the

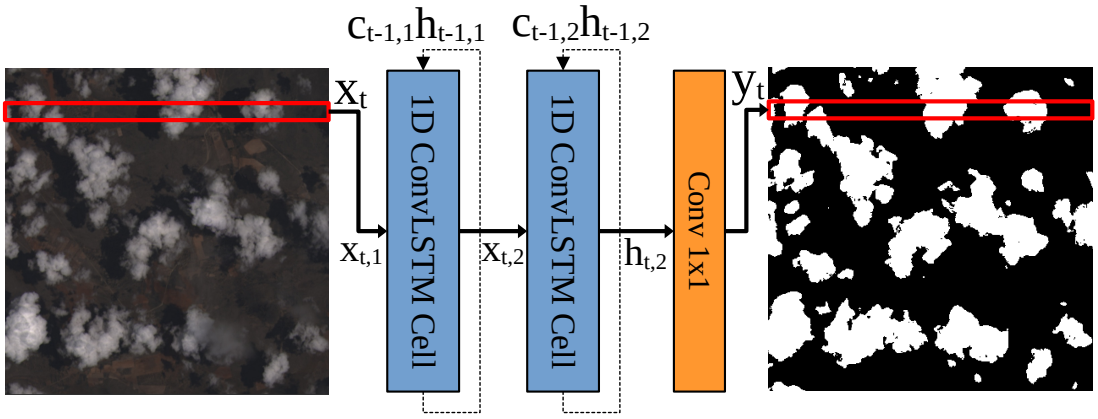


Figure 3.1: Our 1D Convolutional LSTM Neural Network, ScannerNet, segments the input images line-by-line to save memory and reduce latency. We use two 1D ConvLSTM [Xin+15] layers and a single 1x1 convolution to obtain a binary output. Our architecture allows a 280x reduction in memory usage compared to previous works on cloud segmentation at a similar number of parameters.

Network	F_1	F_2	Filter size	Parameters
ScannerNet	4	8	9	4 521
ScannerNet Small	4	4	7	1 717

Table 3.1: Number of filters, filter size and number of parameters for each of our two ConvLSTM networks. F_1 and F_2 are the number of filters in the first and second 1D ConvLSTM layer, respectively.

hidden state of layer 2 $h_{t,2}$ to a single line of the segmentation map, as shown on Figure 3.1.

In order to obtain results for two different total numbers of parameters and thus easily compare our networks to previous works, we create two networks with varying F_1, F_2 and filter size. Table 3.1 shows the details of these two networks.

3.3 Experiments

In order to validate our design, we perform a cloud segmentation experiment. On-board cloud segmentation is an important topic as it can be the first step in the on-board compression of satellite images. Indeed, a large portion of the Earth is covered by clouds at any given time, thus the removal of cloud pixels translates to large savings in both bandwidth and storage space. We compare our results to the lightweight networks presented in Chapter 2.

Network	IoU	F1	Accuracy	Precision	Recall
Baseline	68.14	72.89	83.58	72.92	72.85
C-FCN	82.04	85.77	91.74	89.68	82.19
C-UNet++	**86.18**	**89.75**	93.52	86.19	**93.62**
ScannerNet Small (ours)	81.86	85.51	91.71	90.86	80.75
ScannerNet (ours)	85.80	88.98	**93.65**	**93.79**	84.65

Table 3.2: Quantitative results of our architecture on the 38-Cloud dataset [MS19] at a detection threshold of 0.5, compared to networks from the previous chapter. Our method obtains similar metrics at similar numbers of parameters, while using less memory and having less latency.

For this experiment, we use the 38-Cloud dataset [MS19] and implement our architecture in the PyTorch 1.10 deep learning framework. We re-implement the C-FCN and C-UNet++ networks in this framework, in order to ensure a fair comparison. As done in the previous chapter, we only use the RGB bands of the input images. As a baseline, we also implemented a non-recurrent 1D convolutional network with a similar number of parameters as our architecture. The 38-Cloud dataset is composed of 384x384 pixel crops from Landsat-8 scenes. We use 15% of the training set as a validation set. Using early stopping, all networks are trained until convergence for a maximum of 1000 epochs. We use a batch size of 64 and the Adam optimizer for all networks. Training was done on a machine with an AMD Ryzen 9 3900X CPU, Nvidia GeForce RTX 3090 GPU and 64GB of DDR4 RAM.

3.3.1 Quantitative results

Table 3.2 and Figure 3.3 show quantitative results using commonly used metrics (see definitions in [MS19]). We observe that our ScannerNet Small network obtains results similar to the ones of C-FCN, even though its memory usage is much lower. Our bigger ScannerNet obtains slightly lower scores than C-UNet++, also while consuming less memory for its forward pass. The baseline 1D convolutional network (without LSTM) obtains much worse results, even though it has a similar number of parameters as ScannerNet Small, which shows that the recurrent part of our architecture is necessary to obtain a decent segmentation.

3.3.2 Qualitative results

Figure 3.2 shows a selection of qualitative results taken from the 38-Cloud test set. We observe that even though it can only use one line of input at any step to generate the output segmentation result, our network manages to generate

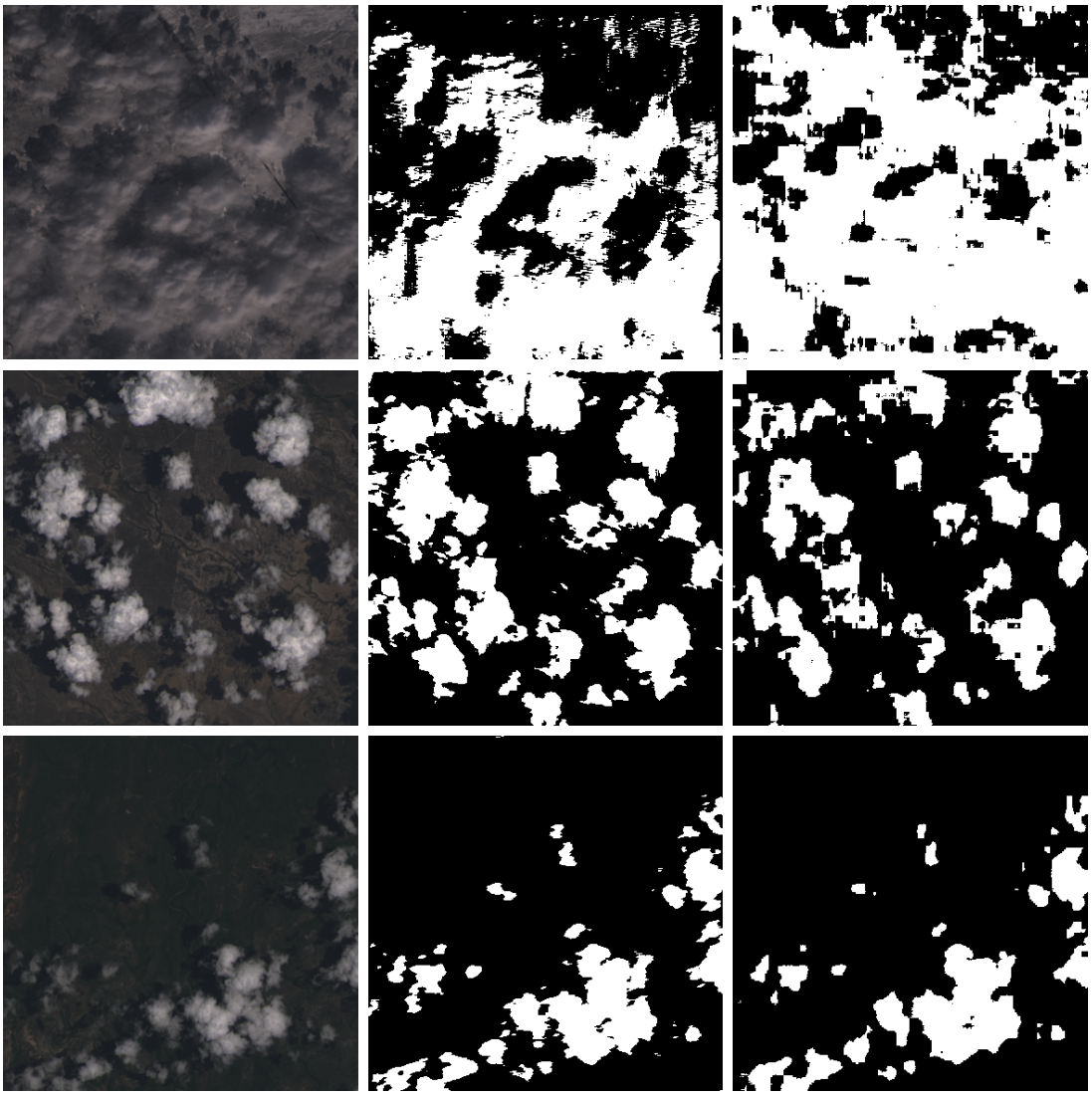


Figure 3.2: Qualitative results of **ScannerNet (ours, middle)** against C-UNet++ (right) on a selection of images from the 38-Cloud test set. Our architecture performs well under a variety of scenarios, including images with snow backgrounds, even though it only processes one line of the input image at a time.

coherent results that match the output of C-UNet++. Moreover, we can observe some block artifacts in the output of C-UNet++ that do not exist in the output of ScannerNet. Our intuition is that since ScannerNet performs all processing at the same resolution as the input image, it can output a more refined segmentation than C-UNet++.

3.3.3 Efficiency

In Table 3.3, we report the number of parameters used by our networks, and the memory required for a single forward pass of each network, as reported by the `torchinfo` module. We observe that our architecture enables some significant memory savings compared to the already memory efficient architectures presented in Chapter 2. Indeed, at a similar number of parameters as C-FCN, we obtain a 280x reduction in memory usage.

Next, we use a low-power edge device to measure the latency of our neural networks. In our context, we define the latency as the delay between the start of the processing and the output of the first pixel of the segmentation map, without including data loading time. For our ScannerNet architecture, since the processing is done line-by-line, the latency is thus measured on the computation of a single line of segmentation. For other neural networks, since the processing is done on the whole image at once, the latency is measured on the computation of the whole segmentation.

As our test system, we use an Nvidia Jetson Nano development kit with 2GB of memory, since it is a very popular platform for embedded systems. It features an SoC equipped with a quad-core A57 ARM CPU and a 128-core Nvidia Maxwell GPU, consuming up to 10W. For benchmarking, we lock the clock of the SoC to its maximum using the `jetson_clocks` utility, and cool the board with a Noctua NF-A4x20 fan running at maximum speed. We run each test 500 times in a row with 10 warmup runs. We use the same software configuration as the one used for training. We test each network on both the CPU and the GPU to take more usage scenarios into account (e.g. systems without GPUs).

The results of this latency testing are shown in Table 3.4. We observe a significant reduction in latency compared to previous works. In particular, our small ScannerNet offers a 3.7x reduction in latency compared to C-FCN, with a similar number of parameters. The improvement is even larger when comparing to C-UNet++. We notice a bigger improvement when running on the CPU than on the GPU. Our intuition is that since GPU architectures are more efficient when asked to process a large amount of data at once, our architecture does not offer as much benefit as on the CPU. Nevertheless, the reduction in latency is still significant.

Network	Parameters	Memory usage (MB)
Baseline	1 409	0.08
C-FCN	1 471	28.62
C-UNet++	9 129	47.19
ScannerNet Small (ours)	1 717	**0.10**
ScannerNet (ours)	4 521	**0.15**

Table 3.3: Number of parameters and memory usage (in MegaBytes) for a forward pass of our architecture, compared to networks presented in the previous chapter. Our networks allow significant memory savings during inference.

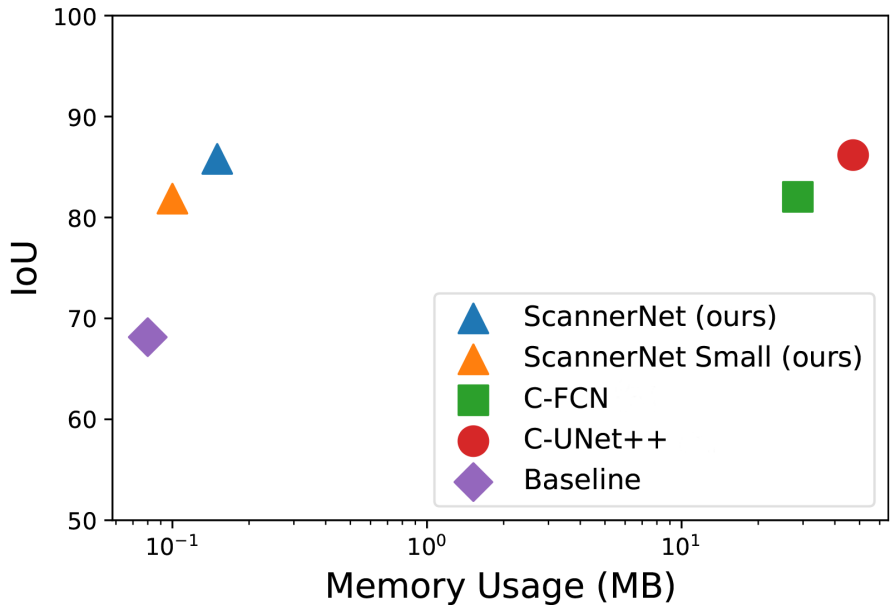


Figure 3.3: IoU (Intersection over Union) of segmentation output at a threshold of 0.5 with respect to memory usage (in MegaBytes, log scale), compared to previous works. Our networks achieve IoU results similar to previous works, while having a comparatively very low memory usage.

Network	Latency (CPU)	Latency (GPU)
C-UNet++	638.6	27.9
C-FCN	296.1	11.1
ScannerNet (ours)	91.5	4.6
ScannerNet Small (ours)	**80.3**	**4.5**

Table 3.4: Average latency (milliseconds, lower is better) of our networks, measured on the CPU and GPU of a Jetson Nano development kit, compared to networks from the previous chapter, on 384x384 pixel images of the 38-Cloud dataset. Our networks provide a 3.7x reduction in latency at the same number of parameters (i.e. when comparing ScannerNet Small to C-FCN).

3.4 Conclusion and future works

In this chapter, we have presented ScannerNet, a 1D Recurrent Convolutional neural network architecture with very low memory usage and latency for on-board segmentation of satellite images. We have shown that our networks obtain results comparable to previous works in a cloud segmentation task while occupying 286 times less memory and having a lower latency at a similar number of parameters. We have observed real-world gains on a low-power device, with a reduction of the latency by a factor of 2.4 in the worst case.

Our approach is orthogonal to previous works on compact architectures and quantization of neural network weights. In particular, our networks could be further optimized by using integer or binary weights. We have also shown that 1D convolutions can work just as well as 2D convolutions for image segmentation, a technique that is seldom explored in the literature. We hope to pave the way for more varied neural network architecture designs adapted to different kinds of on-board sensors. Future work includes testing our architecture on FPGA chips and images from push broom sensors.

Chapter 4

Regression of compact object contours

In this chapter, we introduce an instance segmentation method based on contour regression using Fourier coefficients.

4.1 Introduction

Object detection is the task of finding bounding boxes of objects in images. However, an inherent flaw of bounding boxes is their rectangular shape, which will always limit their IoU (Intersection over Union) with actual underlying objects. This is especially true for objects with holes or multiple sharp angles (*e.g.* animals, bicycles).

Instance segmentation tries to alleviate this problem by providing pixel-level masks for each detected object. Masks, however, also have some drawbacks as an object contour representation: they are resolution-dependent, and have to be resized and interpolated to be used on images with different resolutions. Masks also take up a lot of storage space and require more computational power, which is especially problematic for embedded systems such as drones or satellites. Furthermore, Geographic Information System (GIS) applications like online mapping rely on vector formats, and the automatic conversion of raster masks to more compact representations such as polygons is challenging [LLM20].

A good middle ground between bounding-box regression and mask-based instance segmentation seems to be the regression of object contours using shape encoding, the goal of which is to capture the boundary of objects using a simple parametric function. Methods such as PolarMask [Xie+20] regress the contour of objects directly at a fixed set of sampling points revolving around the center of objects. Some other methods use active contours or snake algorithms [Pen+20].

Finally, methods such as ESE-Seg [Xu+19a] and FourierNet [MBZ20], make use of Chebyshev or Fourier coefficients to explicitly regress an encoding that represents the shape of the object. However, because such methods define object shapes as polar functions, they are limited to the representation of star-shaped domains.

Our work proposes Smooth Contour Regression (SCR), a method that captures resolution-free object contours as complex periodic functions using a Fourier shape decoder. The method offers a good compromise between accuracy and compactness thanks to the design of efficient geometric shape priors. SCR brings several important improvements over previous methods: First, we present mathematical arguments that Fourier coefficients constitute a better representation than the Chebyshev coefficients used in [Xu+19a]. By careful design of the loss function, we improve the visual quality of regressed contours with fewer Fourier coefficients, while also simplifying the training process. Moving to a complex representation for contours allows us to halve the number of calls to the Inverse Fast Fourier Transform (IFFT) in the final shape decoding stage, while allowing more freedom in shape representation by alleviating the limits of polar coordinates-based methods. We benchmark SCR on the popular COCO 2017 instance segmentation dataset, and show its competitiveness against existing algorithms in the field. Finally, we propose a compact version of our SCR architecture based on a lightweight backbone [RF18] for use on low-power systems, typically found on board satellites and UAVs, targeting applications such as real-time on board instance segmentation and efficient transmission within constellations of satellites. We benchmark this compact version on embedded hardware with a wide range of power targets, achieving up to real-time performance.

4.2 Related Work

In this section, we go over some related works in object detection, instance segmentation, and contour regression.

4.2.1 Detection neural networks

R-CNN [Gir+14] and its successors [Gir15; Ren+15], were the original neural networks for object detection. They are two-stage approaches, whereby object locations are first proposed by a Region Proposal Network (stage one) and then classified by a subsequent CNN (stage two). Later, fully convolutional single-stage approaches were proposed [Liu+16; Lin+17b; Red+16], which were significantly faster than their two-stage counterparts. These methods rely on a set of "anchor" bounding boxes, to which the detected boxes are assigned and regressed relatively. The simplicity and efficiency of these single-stage detectors have made

them very popular over the years. Many improvements to this design have been proposed, such as the Feature Pyramid Network (FPN) [Lin+17a] and subsequent work [TL19; GLL19; Wan+19a], which aggregate features from the backbone at different levels to create semantically rich feature maps at each level. Recently, FCOS [Tia+19], an anchor-free detection neural network architecture, has been developed. By leveraging the pyramid structure given by the FPN, FCOS removes the problem of anchor box assignment, and thus streamlines the object detection task with a one-stage, proposal-free, anchor-free framework.

4.2.2 Mask-based instance segmentation neural networks

Instance segmentation neural networks were pioneered by Mask-RCNN [He+17], a two-stage approach based on [Gir+14]. A significant number of architectures have been developed [Wan+20c; Wan+20b; Che+19a; Liu+18; Cao+19], each bringing incremental improvements in popular benchmarks. Some methods save storage space by regressing masks as low-dimensional embeddings; these are decoded using either a learned decoder [Jet+17; Zha+20], which complexifies the model and its training, or fixed functions such as the Discrete Cosine Transform [She+21], in which case the encoding dimension is still quite large. Nonetheless, mask-based instance segmentation methods all encounter the drawbacks of pixel-based representation, and are generally much slower than their bounding box-based counterparts.

4.2.3 Contour regression methods

Recently, the community has explored new ways of regressing the boundaries of objects. These methods aim to alleviate the aforementioned issues of pixel-based mask representation while offering better IoU with detected objects than bounding box-based architectures, which most take as a starting point.

Snake-based methods [Pen+20; Liu+21] represent shapes as polygons in pixel coordinates and deform them iteratively using circular convolutions. While these methods can reach good accuracy levels at reasonable speeds, they generally use a high number of contour points, which can be costly in terms of storage on embedded devices, especially compared to the shape encoding methods described below.

ESE-Seg [Xu+19a] is based on the YOLOv3 detection neural network [RF18] and regresses an explicit shape encoding for each detected object in the form of either Chebyshev or Fourier coefficients and a center. The contour of the object is modeled as a function $\rho(\theta), \theta \in [0, 2\pi]$, revolving around this center, as shown in the left part of Figure 4.1. Corresponding ground truth coefficients have to be generated for each object of the training dataset. However, when used with

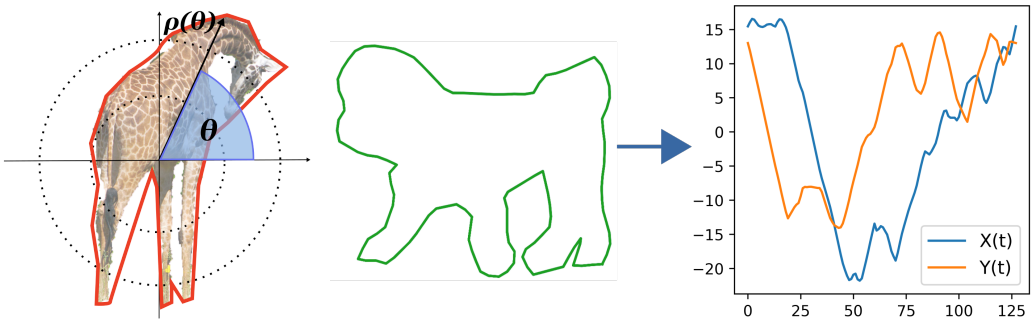


Figure 4.1: **Left:** Polar contour representation, where the shape is decribed as a single 2π-periodic function $\rho(\theta)$. **Right:** Cartesian contour representation, where the shape of the object is described by two periodic functions: $X(t)$ and $Y(t)$.

Chebyshev coefficients, this representation is not well suited to periodic functions, as we will demonstrate later. Moreover, this method is only able to fully represent star-shaped domains and thus cannot be used to regress less regular shapes.

In PolarMask [Xie+20], the authors modify the FCOS [Tia+19] architecture in order to regress the polar coordinates of the contour points directly, *i.e.* the values of the function ρ. Again, specific ground truth labels have to be created for each object, with a fixed number of rays cast from its center, and only star-shaped objects can be regressed, since only one coordinate is regressed for each ray.

FourierNet [MBZ20], also based on FCOS, simplifies the contour regression problem by using the Inverse Fast Fourier Transform (IFFT) as a differentiable shape decoder. Thus, the neural network regresses Fourier coefficients, but the output is directly compared to ground truth polygons. The authors use either polar or Cartesian coordinates. In the polar case, one set of complex coefficients is used to find ρ, and the regressed shapes have to be star-shaped. In the Cartesian case, the contour is seen as two functions, $X(t)$ and $Y(t)$, as shown in Figure 4.1. Two sets of complex coefficients are used (one for each) to model these functions. While the authors manage to get good results in the polar case, their Cartesian version does not perform as well quantitatively and fails to produce coherent shapes when using a high number of Fourier coefficients.

4.3 Method

Our SCR network architecture is based on the FCOS detection neural network [Tia+19]. Like most recent detection architectures, FCOS is composed of a backbone, a neck (feature pyramid), and several detection heads with shared weights, as shown in Figure 4.2. As backbones, we use ResNet-50 [He+16], ResNeXt-101

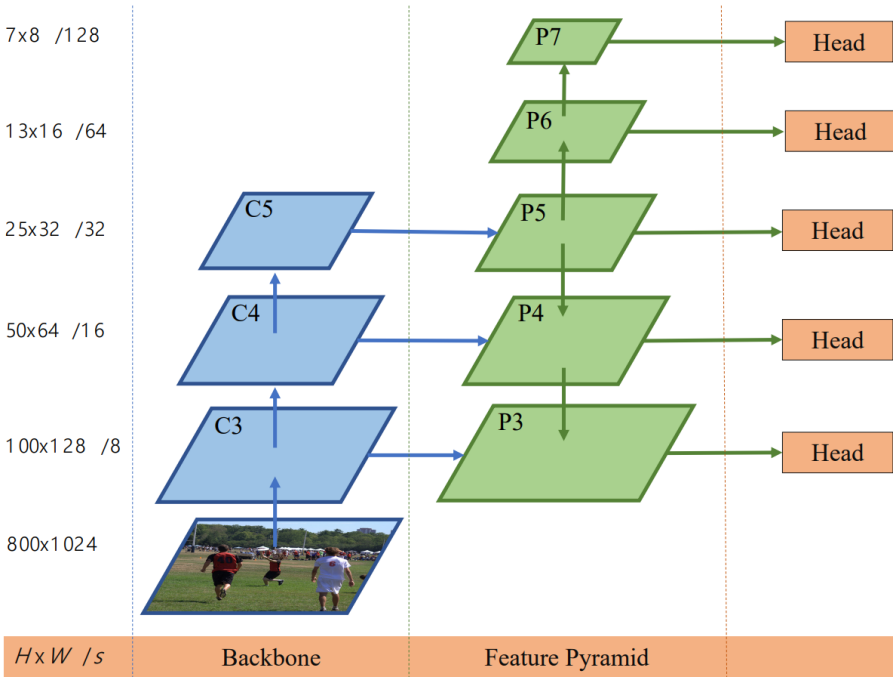


Figure 4.2: FCOS Architecture [Tia+19], which we use as a starting point for our SCR method. Figure from [Tia+19].

[Xie+17], and the lightweight DarkNet-53 [RF18] to address different performance targets. We run experiments with both the classical FPN neck from [Lin+17a], and the more recent FPN-CARAFE [Wan+19a], which provides a small accuracy improvement.

4.3.1 Choice of shape representation

In [MBZ20; Xu+19a; Xie+20], the shape of an object is represented as a function $\rho(\theta)$ with $\theta \in [0, 2\pi]$, and thus revolves around a center, which is either regressed along with the coefficients in [Xu+19a], or is the center of output "cells" from FCOS in [MBZ20; Xie+20]. ρ is a 2π-periodic function measuring the distance of each point of a contour to this center. By contrast, we choose to model the shape of objects with a more generic complex periodic function of the form $C(t) = X(t) + iY(t)$ with $t \in [0, 1]$.

4.3.2 On Chebyshev coefficients

In this section, we demonstrate why Chebyshev coefficients, as used in [Xu+19a], are not well suited to the regression of continuous periodic functions through a

neural network – in this case, the ρ function.

Chebyshev polynomials are obtained through the following recurrence relation:

$$\begin{aligned} T_0(x) &= 1 \\ T_1(x) &= x \\ T_{n+1}(x) &= 2x\,T_n(x) - T_{n-1}(x). \end{aligned} \tag{4.1}$$

Let N be the number of degrees used for interpolation of ρ using the truncated Chebyshev series. Then for $x \in [-1, 1]$ we have:

$$\rho(x) = \sum_{n=0}^{N} \alpha_n T_n(x) \tag{4.2}$$

Since the contour ρ is a continuous periodic function on $[-1, 1]$, in order to perfectly interpolate it, the Chebyshev coefficients α_i must satisfy:

$$\sum_{n=0}^{N} \alpha_n T_n(-1) = \sum_{n=0}^{N} \alpha_n T_n(1) \tag{4.3}$$

We also know that Chebyshev polynomials satisfy the following properties, for all $n \in \mathbb{N}$:

$$\begin{aligned} T_n(1) &= 1 \\ T_{2n}(-1) &= 1 \\ T_{2n+1}(-1) &= -1 \end{aligned} \tag{4.4}$$

Then, we have the following:

$$\sum_{k=0}^{N/2} \alpha_{2k} - \sum_{k=0}^{N/2} \alpha_{2k+1} = \sum_{n=0}^{N} \alpha_n \tag{4.5}$$

Thus, we have shown that in order to perfectly interpolate a continuous periodic function, the Chebyshev coefficients must satisfy the following relationship:

$$\sum_{k=0}^{N/2} \alpha_{2k+1} = 0 \tag{4.6}$$

It is unlikely that a neural network can learn this exact relationship, even if optimized as a loss during training. Hence, discontinuities can be observed in nearly every shape regressed in this manner. Figure 4.3 shows the discontinuities created when this relationship is not strictly enforced, e.g. when the Chebyshev coefficients are regressed by a neural network. They appear at $\theta = 2\pi$ on every regressed shape.

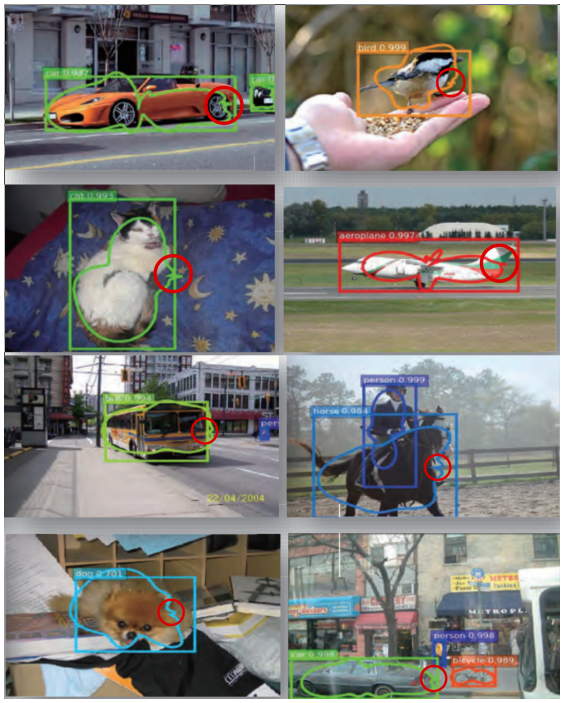


Figure 4.3: Discontinuities found in the results of [Xu+19a] when using Chebyshev coefficients for shape regression, circled in red.

4.3.3 Contours as a complex function and Fourier series

The aforementioned drawback of Chebyshev polynomials steers us toward the Fourier decomposition, which is inherently better suited to our purposes, since any set of Fourier coefficients will always represent a continuous smooth periodic function.

Our single complex function representation has several advantages over using two real functions as in [MBZ20]: not only does it halve the number of IFFT calls in the shape decoding stage, but it also ties these coefficients together as they represent a single function, which might help during the training of the neural network, as backpropagation occurs through a single IFFT call.

In order to evaluate how many complex coefficients are needed to represent shapes accurately, we interpolate all polygons of the COCO 2017 validation set using the scheme detailed in section 4.3.8. We then compute the FFT and its inverse after zeroing a certain number of coefficients. Finally, we compare the result to the original polygons using the Chamfer Distance as reconstruction error. Figure 4.4 shows the quickly diminishing returns of increasing the number of coefficients, highlighting the fact that a small number suffices to represent a shape accurately. We decided to run our experiments using 8 to 32 complex coefficients.

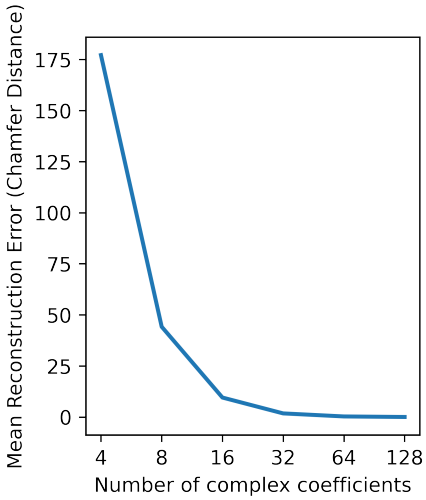


Figure 4.4: Mean Reconstruction Error (Chamfer Distance) computed over the COCO 2017 validation set for a varying number of complex Fourier coefficients, showing the rapidly diminishing returns of increasing the number of Fourier coefficients. The polygons are interpolated to 128 points.

4.3.4 SCR head

The SCR head, shown in Figure 4.5, is based on the FourierNet [MBZ20] head. For both the class and contour regression branches, we use 3 deformable convolutions [Zhu+19] with shared weights among feature levels. Thanks to the complex contour representation, we only require one call to the IFFT for shape decoding. We add two regularization losses, which we will elaborate after introducing the loss function, below. The first one is applied directly to the regressed coefficients. The second one is applied to the decoded point coordinates before they are scaled according to the stride of the feature level and a learnable scaling coefficient.

4.3.5 Loss function

For training SCR, our loss function is defined as a sum of different terms:

$$\mathcal{L} = \mathcal{L}_{\text{CD}} + \mathcal{L}_{\text{cent}} + \mathcal{L}_{\text{cls}} + \mathcal{L}_{\text{perim}} + \mathcal{L}_{\text{coeff}} \tag{4.7}$$

where $\mathcal{L}_{\text{CD}}$ is a polygon regression loss based on a symmetrized Chamfer Distance (similar to [MBZ20]), $\mathcal{L}_{\text{cent}}$ is the centerness loss from FCOS [Tia+19] (binary cross-entropy), and $\mathcal{L}_{\text{cls}}$ is the Focal Loss for classification from RetinaNet [Lin+17b]. $\mathcal{L}_{\text{perim}}$ and $\mathcal{L}_{\text{coeff}}$ are regularization terms on the shape perimeter and the Fourier coefficients, respectively. Both are detailed below.

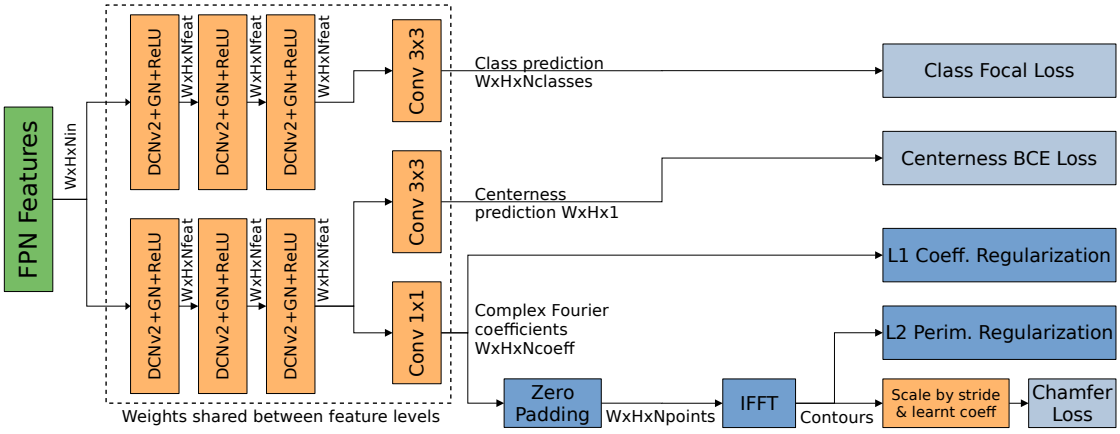


Figure 4.5: Our SCR head, which is applied to each output feature level of the FPN. Learned operators are in orange. Fixed operators are in blue. One branch of the head is used for the class prediction. The other branch is used for both the centerness prediction and the Fourier coefficient regression. All weights are shared between feature levels except the final scaling coefficient. An L^1 regularization is applied to the Fourier coefficients to direct training towards the regression of simpler shapes. An L^2 regularization is used on the perimeter of the regressed contours to encourage the regression of shapes that circle the object only once.

Note that, in contrast to [MBZ20], we do not use a bounding box loss function because bounding boxes can be inferred from the regressed polygons and do not need to be learned separately.

4.3.6 L2 Perimeter regularization

Using the Chamfer distance loss as the main polygon regression loss has a major drawback that needs to be addressed: since ground truth points are only compared to the closest regressed point, the network does not necessarily learn to regress contours that go around the object only once. Indeed, circling the object multiple times is not penalized in $\mathcal{L}_{\text{CD}}$. One way to alleviate this problem, as done in [MBZ20], is to first perform a warm-up of the shape regression with an L^1 loss, so that each output point is already tied to specific ground truth points. However, after being trained with Chamfer loss, their network still tries to regress shapes in an overly complex way, which creates many self-intersections in the output polygon, as seen in our comparison in section 4.4 (Figure 4.12).

We instead choose to apply a L^2 perimeter-based regularization to every output contour, even the ones that have not been assigned to any object. In addition to simplifying the training process, this approach encourages simpler shapes that go around the object once and have a minimal number of self-intersections:

$$\mathcal{L}_{\text{perim}} = \lambda_{\text{perim}} \sqrt{\sum_{i=0}^{N} (\mathbf{x}_i - \mathbf{x}_{i+1})^2} \tag{4.8}$$

The λ_{perim} coefficient can be set to a relatively low number to mitigate its influence on training. Setting it too high will result in small round-ish shapes. It can be decayed over time or set to zero after $\mathcal{L}_{\text{perim}}$ reaches a steady state to retrieve more complex shapes while retaining the effect of regularization, as shown in our experimental results (Figure 4.11).

4.3.7 L1 Fourier coefficient regularization

While representing contours using Fourier coefficients is already more storage-efficient than using masks, we may want to further reduce the storage space needs by ignoring very small coefficients (*i.e.* setting them to zero after regression). Thus, we include an option for enforcing sparsity to bring as many coefficients as possible close to zero.

In order to favor sparsity of the Fourier coefficients F_i, we use an L^1 regularization term:

$$\mathcal{L}_{\text{coeff}} = \frac{\lambda_{\text{coeff}}}{N_c} \sum_{i=-n, i \notin \{-1,0,1\}}^{n} |F_i| \tag{4.9}$$

Where λ_{coeff} is an adjustable parameter (500.0 in our experiments). Note that we do not apply this penalty to the -1, 0, and 1 frequency coefficients, since these coefficients will have to be quite large in most cases. Having many Fourier coefficients close to zero, especially high- frequency ones, also helps the network create simpler and smoother contours, as shown in section 4.4 (Figure 4.13).

4.3.8 Polygon ground truth processing

Existing works [MBZ20; Xie+20; Xu+19a] typically convert the ground truth polygons to polar coordinates with respect to a center. This is done by casting a number of rays at regular angular intervals from this center to the farthest contour point. However, this process is difficult to implement on-the-fly without slowing the data loading pipeline. We also note that the distance between two adjacent points created in this manner can vary greatly (*e.g.* in oblong objects). We believe that having a high number of evenly spaced ground truth points is better for the stability of the Chamfer Distance loss. Thus, our interpolation is done in a "constant spacing" fashion for each ground truth polygon. Since the original ground truth polygons vary in number of points, we interpolate them such that

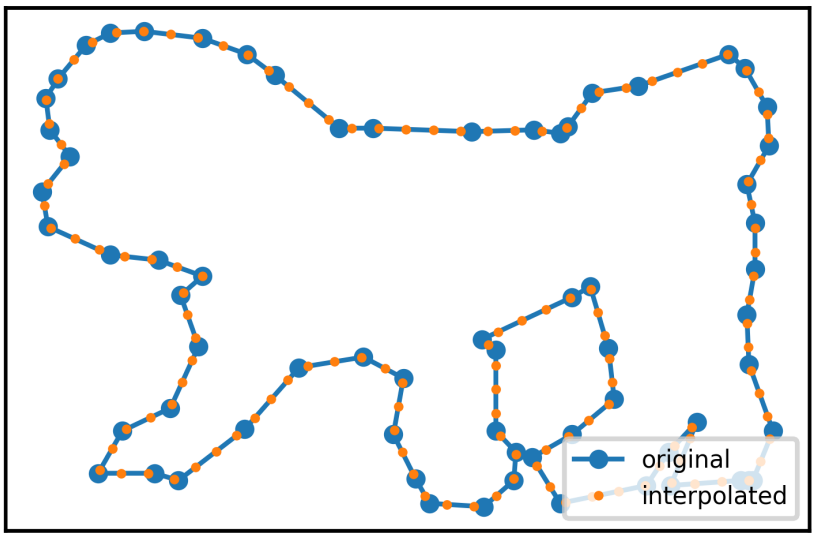


Figure 4.6: Example of polygon interpolation done on-the-fly during data loading.

the number of ground truth points N_p matches the number of regressed points. Our approach is easily done on the fly in the data loading pipeline and does not incur any performance loss.

4.4 Experiments

In this section, we compare our method to state-of-the-art contour regression and instance segmentation methods on the popular COCO 2017 dataset [Lin+14].

4.4.1 Implementation details

We implement our ResNet-50, ResNeXt-101 and DarkNet-53 based models using the MMDetection 2.7.0 object detection toolbox [Che+19b], based on MMCV 1.2.1 for easy and fair comparison with other methods. The underlying framework is PyTorch 1.7.0 with CUDA 10.1. We use the updated FFT package from PyTorch 1.7.0, which allows the use of complex numbers. We use the Chamfer Distance implementation from PyTorch3D 0.3.0 [Rav+20]. The interpolation scheme described in 4.3.8 is implemented as a module in the MMDetection data loading pipeline using the `interp` module from Numpy. Notably, due to software incompatibilities during the conversion of our DarkNet-53 model for low-power hardware targets, we had to simplify the architecture by removing deformable convolutions

Name-Backbone	DCNv2	CARAFE	N_{in}	N_{feat}	Training time
SCR-D53	No	No	128	128	20h
SCR-R50	Yes	Yes	256	256	40h
SCR-X101	Yes	Yes	256	256	80h

Table 4.1: Characteristics of our main networks. Variations are further explored in our ablation study. DCN refers to the use of Deformable Convolution v2 [Zhu+19] layers in the head. CARAFE refers to the use of the FPN-CARAFE [Wan+19a] neck. Approximate training time reported on 8 Tesla V100 GPUs.

and CARAFE. In order to evaluate our method and easily compare it to previous works, we chose to work on the widely used COCO 2017 Instance Segmentation dataset [Lin+14].

Our networks are trained for 38 epochs using SGD with momentum (0.9), with an initial learning rate of 0.0003, batch-size of 16 (2 images per GPU), gradient clipping at a maximum L^2 norm of 45, and a 500-step warm-up with a ratio of 0.001 applied to the learning rate. We use the multiscale training feature found in MMDetection to train with image heights of 640 and 800 pixels. We balanced our loss function terms through trial-and-error, and settled on the following values: λ_{cls}=1.0, λ_{cent}=1.0, λ_{CD}=1.0, λ_{perim}=0.01, λ_{coeff}=500.0.

4.4.2 Qualitative results

Figure 5.1 shows a selection of results from the COCO 2017 test-dev dataset, computed using our smallest DarkNet-53 based network. Our method properly regresses smooth artifact-free contours in scenes with varying numbers of objects, classes, and shapes. In particular, non-star-shaped objects can be accurately represented. Figures 4.9 and 4.8 show an extended selection of qualitative results.

Figure 4.10 shows a comparison of our method against ESE-Seg [Xie+20], which is based on Chebyshev coefficients. We observe that our method regresses smooth continuous shapes that properly follow the objects thanks to the Fourier representation.

4.4.3 Quantitative comparison

We compare our method to existing shape encoding methods [MBZ20; Xu+19a; Xie+20], as well as state-of-the-art snake-based [Pen+20; Liu+21] and mask-based methods [He+17; Liu+18; Wan+20c]. The evaluation uses three metrics:

1. mAP for accuracy (as defined in [Lin+14])

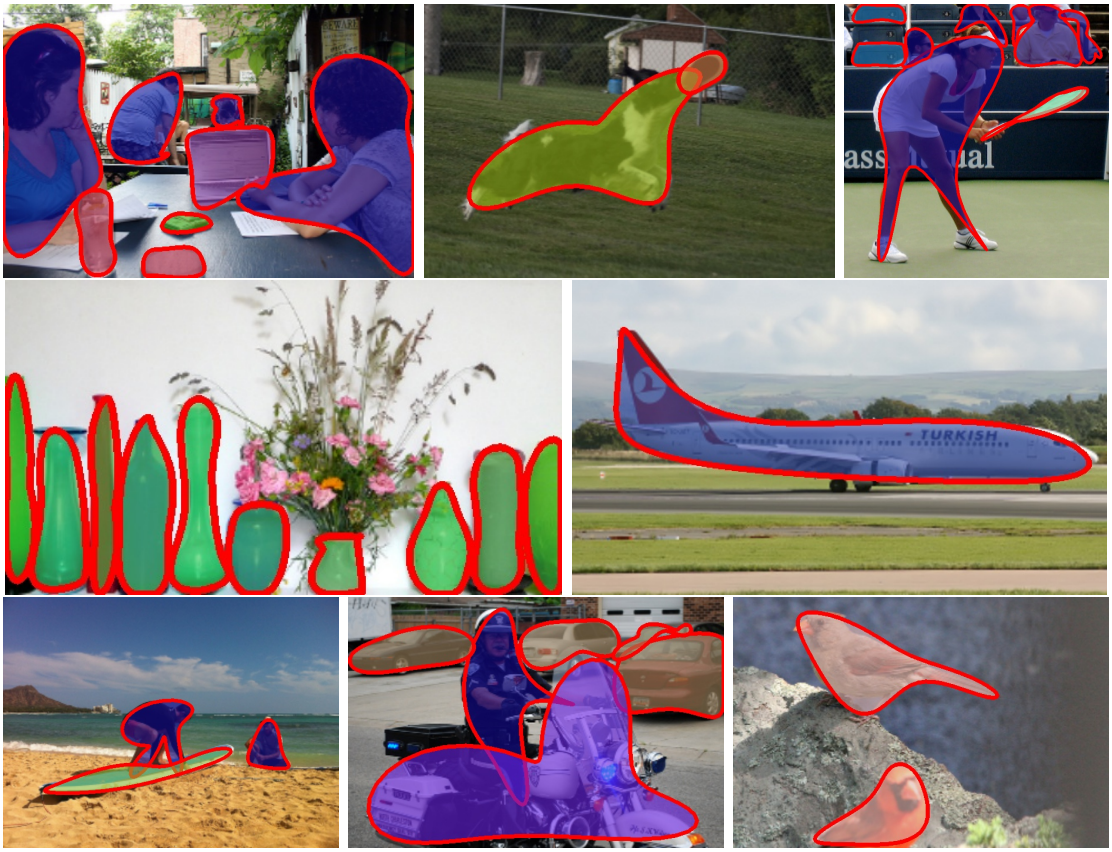


Figure 4.7: Our Smooth Contour Regression (SCR) algorithm captures the silhouette of objects with a compact resolution-independent representation based on Fourier coefficients. Contours produced by our algorithm adequately approximate the general shape of free-form objects such as persons, animals, cars, or pots while being defined in a simple parametric way with only a few complex Fourier coefficients, here 8. Images from COCO 2017 test-dev.

2. FPS (frames per second) reported on a single Nvidia GTX 1080Ti GPU for speed

3. Memory usage per regressed object in bits, which we call *SEC* (for Shape Encoding Complexity).

Run time and object size in memory or storage are important factors for embedded hardware applications, such as real-time on-board change detection. Thus, we propose an OES (Overall Efficiency Score), which is simply the multiplication of these three scores and is thus defined as $OES = 100 \times mAP \times FPS \times SEC^{-1}$. This metric represents the trade-off between having slow but accurate models outputting detailed shapes, and having faster models outputting simpler shapes.

Figure 4.8: More results from our DarkNet-53 based model selected from COCO 2017 test-dev.

Figure 4.9: More results from our DarkNet-53 based model selected from COCO 2017 test-dev.

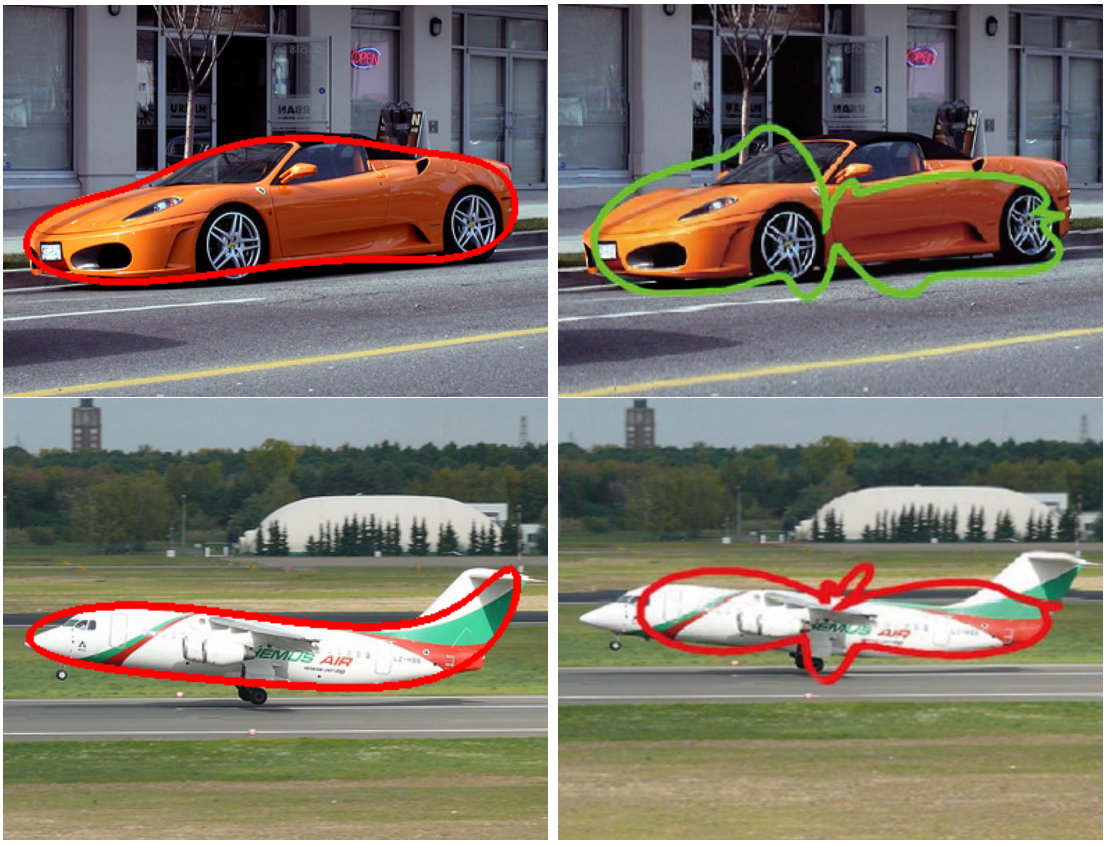


Figure 4.10: Our SCR-D53 result (left) compared to ESE-Seg [Xie+20] result (right) with the same backbone on an image from the Pascal VOC2012 dataset. The advantage of the Fourier representation is apparent, as our method regresses a smooth continuous shape.

The comparisons against other shape encoding methods are shown in Table 4.2 for the ResNeXt-101 backbone and Table 4.3 for the DarkNet-53 backbone. We observe that with the same backbone and at a similar number of coefficients, our method achieves good accuracy compared to other shape encoding methods.

In Table 4.2, our method has a competitive efficiency score (OES). While [MBZ20] is slightly faster, our method achieves better accuracy. By contrast, PolarMask [Xie+20] can achieve higher mAP, but it does so at the cost of a higher number of coefficients, and thus has lower overall efficiency.

Table 4.3 shows that our method is faster and more efficient than ESE-Seg [Xu+19a]. In addition, even though its mAP score is slightly lower, our method does not suffer from the visual drawbacks of Chebyshev coefficients detailed in section 4.3.2.

The comparison against other types of approaches is shown in Table 4.4. Snake-

Method	N_{coeff}	mAP	FPS	SEC	OES
PolarMask [Xie+20]	36	**32.9**	4.1	1152	11.7
FourierNet-Cartesian [MBZ20]	16	22.9	**4.9**	**512**	21.9
FourierNet [MBZ20]	16	23.3	**4.9**	**512**	22.3
Ours (SCR-RX101)	16	27.3	4.2	**512**	**22.4**

Table 4.2: COCO 2017 test-dev results compared to other shape encoding methods based on ResNeXt-101 with an image height of 800 pixels. N_{coeff} in real numbers for shape encodings: one complex coefficient counts as two. SEC: size of a single detected object in memory (bits). OES: Overall Efficiency Score.

Method	N_{coeff}	mAP	FPS	SEC	OES
ESE-Seg[Xu+19a]	20	**21.6**	38.5	640	130
Ours (SCR-D53)	**16**	21.2	**39.1**	**512**	**162**

Table 4.3: COCO 2017 val results for models based on DarkNet-53 with an image height of 416 pixels. N_{coeff} in real numbers for shape encodings: one complex coefficient counts as two. SEC (Shape Encoding Complexity): size of a single detected object in memory (bits). OES: Overall Efficiency Score.

based methods typically offer better accuracy than shape encoding methods; however, their overall efficiency suffers from a high shape encoding complexity, and they are slower than our method when using the same processing size and backbone. Mask-based methods reach very high mAP scores but are also hindered by their speed and SEC and thus generally have a lower overall efficiency.

While our method does not necessarily give the best score for each evaluation metric, it offers a good compromise between them. In particular, the OES scores obtained by our method are competitive compared to existing methods under the same backbone.

4.4.4 Effect of perimeter penalty decay

As discussed in section 4.3.6, it is possible to set λ_{perim} to zero after $\mathcal{L}_{\text{perim}}$ stabilizes. Removing the perimeter regularization after a few iterations (2000 here) yields more complex and accurate contours while retaining the effect of regularization. In Figure 4.11, the network learns to properly follow the neck of the giraffe (left), which is not the case without perimeter penalty decay (right). Setting $\lambda_{\text{perim}} = 0$ right away, *i.e.* removing $\mathcal{L}_{\text{perim}}$ entirely, will let the network circle objects multiple times, leading to poor results, as shown in our ablation study and in Figure 4.13(a).

	Method	N_{coeff}	mAP	FPS	SEC	OES
Snake	DeepSnake [Pen+20]	256	31.0	6.68*	4096	5
	DANCE [Liu+21]	196	34.6	7.6	3136	8
Mask	Mask R-CNN [He+17]	784	33.6	10.8	784	46
	PANet [Liu+18]	784	38.2	4.5	784	22
	SOLOv2 [Wan+20c]	64000	**38.8**	**12.4**	64000	0.75
	Ours (SCR-R50)	**16**	24.2	10.7	**512**	**51**

Table 4.4: COCO 2017 test-dev results compared to mask and snake-based methods with a ResNet-50 backbone and an image height of 800 pixels. N_{coeff} in real numbers for shape encodings: one complex coefficient counts as two. SEC: size of a single detected object in memory (bits). For snake-based methods, we assume pixel coordinates are stored as 16-bit integers. For mask-based methods, we assume masks are binary. OES: Overall Efficiency Score. *: Number extrapolated from [Liu+21]

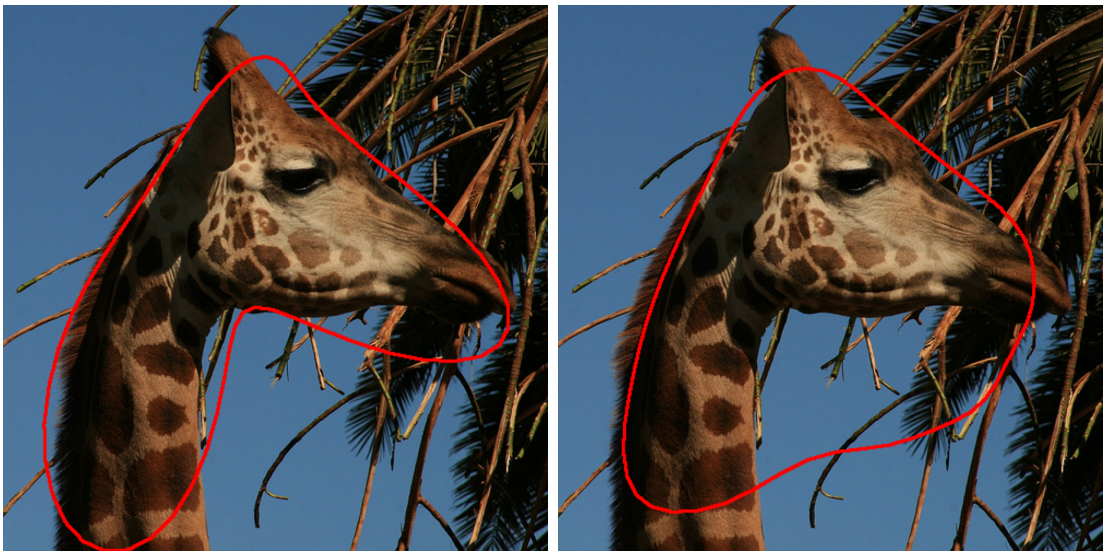
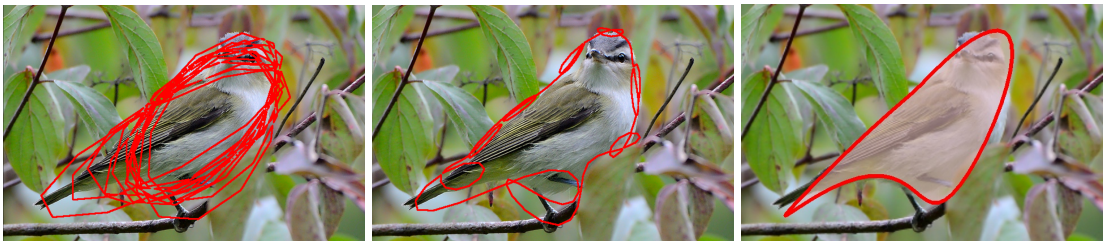


Figure 4.11: SCR-D53 result with (left) and without perimeter penalty decay (right). The perimeter regularization loss coefficient λ_{perim} is set to zero once $\mathcal{L}_{\text{perim}}$ reaches a steady state (2000 iterations here), which yields a more detailed shape while retaining the effect of regularization.

Figure 4.12: Thanks to our coefficient regularization, our method **(left)** learns to regress smooth shapes, whereas the cartesian version of [MBZ20] **(right)** regresses erratic shapes when using a high number of coefficients (here 32).



(a) Perim. reg. disabled (b) Coeff. reg. disabled (c) Both reg. enabled

Figure 4.13: Disabling our perimeter regularization (a) lets the network circle the object multiple times, while disabling the coefficient regularization (b) leads to erratic shapes with many self-intersections. When both regularizers are enabled (c), the network outputs smooth and accurate shapes with a low number of self-intersections, here zero.

4.4.5 Fourier coefficient regularization

The benefits of our coefficient regularization can be observed in Figure 4.12, where our network (left) learned to regress smooth contours while the Cartesian version of FourierNet [MBZ20] (right) regressed erratic shapes even when using high number of coefficients. Figure 4.13(b) shows that disabling our regularization leads to the same kind of behavior shown in [MBZ20].

4.4.6 Ablation study

We now evaluate the impact of our design choices through an ablation study. The results are summarized in Table 4.5. Numbers in parentheses in the following paragraph refer to corresponding lines of the Table.

- **(#2,#3)** We trained the ResNet-50 based model with 60 and 128 contour points and found that for this set of hyper-parameters, there was a performance degradation when using 128 points. We thus kept this parameter at 60, a finding consistent with the literature.
- **(#1,#3)** We show that the new FPN-CARAFE from [Wan+19a] improves the scores of our method compared to the vanilla FPN, for which it is a drop-in replacement.
- **(#3,#4,#5; #6,#7; #14,#15)** Increasing the number of complex coefficients from 10 (20 real values) to 20 made no significant difference, while increasing it to 32 yielded a small performance increase of 1.4 mAP. Depending on the requirements of the application, this increase in detail might be worth the extra storage space. Note that it is still possible to regress a high number of coefficients, and then decide to set some of them to zero after the fact.
- **(#6,#7,#8,#9)** We find that the models used on smaller image resolutions do not perform very well on small objects (AP_S).
- **(#14,#15)** Using a larger ResNeXt-101 backbone yields a small accuracy improvement, but not to the point where we think it is worth the decrease in FPS and the (up to) two-fold increase in training times. **(#10)** DarkNet-53, on the other hand, seems to perform as well as ResNet-50 at high resolutions, especially on larger objects.
- **(#2,#11,#12,#13)** Disabling the L_1 coefficient penalty leads to a very significant drop in mAP. This outcome is probably due to the erratic shapes regressed without it, as shown in Figure 4.13. Disabling the L_2 perimeter

Exp.	Back.	Neck	Head	N_{coeff}	N_{pts}	H	PR	CR	mAP	AP_{50}	AP_{75}	AP_S	AP_M	AP_L
#1	R50	FPN	DCN	20	60	800	✓	✓	22.4	43.0	21.0	10.8	24.2	29.7
#2	R50	CARAFE	DCN	20	128	800	✓	✓	21.4	44.2	19.0	10.9	22.9	28.4
#3	R50	CARAFE	DCN	20	60	800	✓	✓	23.8	45.0	22.4	11.6	25.6	31.9
#4				40					23.8	46.4	21.8	11.7	25.4	32.3
#5				64					25.2	48.1	23.6	13.1	26.9	33.6
#6	R50	CARAFE	DCN	20	60	360	✓	✓	19.1	36.2	18.1	4.0	19.9	31.3
#7	R50	CARAFE	DCN	64	60	360	✓	✓	18.9	37.3	17.0	4.3	19.5	30.9
#8	D53	FPN	-	16	60	320	✓	✓	17.6	33.8	16.4	1.5	17.0	32.1
#9						416			21.2	39.3	20.4	3.7	22.6	35.5
#10						608			23.9	42.2	24.0	6.2	27.5	36.3
#11	R50	CARAFE	DCN	20	128	800	✓	✗	10.8	22.6	9.3	6.3	12.5	14.2
#12							✗	✓	5.8	14.5	3.5	3.1	6.6	7.8
#13							✗	✗	5.4	13.9	3.1	2.9	6.1	7.3
#14	RX101	CARAFE	DCN	20	60	800	✓	✓	**27.2**	50.3	**26.4**	**14.0**	29.4	35.4
#15	RX101	CARAFE	DCN	64	60	800	✓	✓	**27.2**	**50.9**	26.1	13.9	**29.5**	**35.9**

Table 4.5: Impact of different design choices on our models on COCO test-dev results. N_{coeff} in real numbers (one complex coefficient counts as two). N_{pts} = number of contour points in ground truth and after IFFT, DCN = Deformable Convolution v2, Back = backbone, CR = Coefficient Regularization, PR = Perimeter Regularization, H = image height.

penalty also affects the scores significantly, as the contours circle the objects multiple times and thus miss a lot of detail.

4.4.7 Limitations

Our regularization terms tend to direct the training of our network towards simple shapes without actively preventing self-intersections. Thus, such errors might appear occasionally. In figure 4.14, for instance, we see that while the two leftmost horse contours seem properly regressed, they each present one self-intersection. Nevertheless, in this particular case, the masks given by these contours are still usable. One other limitation is that by design, our method is more tailored towards free-form objects than the regression of piece-wise linear shapes, for which polygon-based methods are better suited.

4.4.8 Performance benchmarks on low-power hardware

We evaluated the speed of our fastest DarkNet-53 based SCR network on a variety of devices, ranging from powerful discrete GPUs to low-power edge devices (Table 5.2).

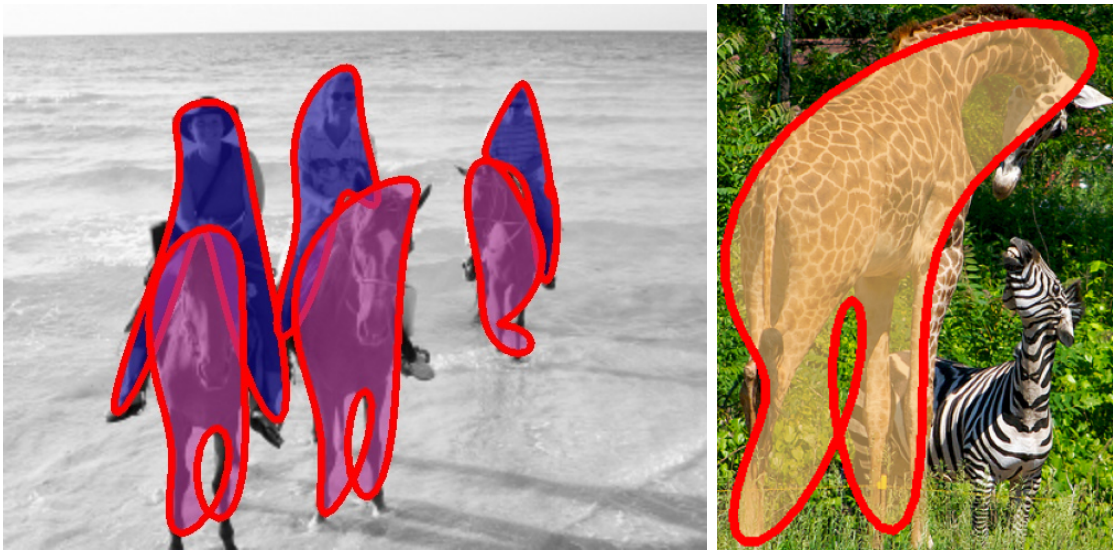


Figure 4.14: Examples of self-intersections that can occasionally appear as they are not actively prevented. The contours still follow the shape of the animals and are thus still somewhat usable.

Our first test platform was the Nvidia Jetson Nano, a popular, affordable, widely available, and reasonably powerful family of development kits. Our method runs at 9.5 FPS on these boards, which is close enough to real-time to be usable in many applications. We also ran the same benchmarks on the more powerful Jetson Xavier NX, featuring INT8 compute capabilities, thus yielding a significant performance boost by bringing the throughput up to 115 FPS. It achieved 5.75 FPS/Watt, making it by far the most efficient combination. The Intel Movidius Myriad X VPU is an ultra low-power AI accelerator with a power consumption of only 1W. The Myriad VPUs have been used in ESA's ϕ-sat-1 CubeSat, as well as in smart security cameras, UAVs and industrial machine vision equipment. We benchmarked SCR on the Myriad X using the imposed OpenVino toolkit and achieved 2.8 FPS, which is very satisfactory for such a low-power device. As a point of comparison with other works, we also ran SCR on an Nvidia GeForce GTX 1080Ti GPU, which is still one of the most popular discrete GPUs. For the sake of completeness, we also provide numbers on an Intel Xeon Silver 4114 CPU.

	Device	Power	Net.	H	Tool	Prec.	FPS	fps/W
Low-power	Nano	10W	Ours	320	TensorRT	FP32	5.4	0.54
					TensorRT	FP16	9.5	0.95
	Nano	10W	Ours	608	TensorRT	FP32	1.72	0.17
					TensorRT	FP16	3.04	0.30
	Xavier NX	30W	Ours	320	TensorRT	FP32	17.2	0.57
					TensorRT	FP16	66.7	2.22
					TensorRT	INT8	**115**	**5.75**
	Myriad X	1.5W	Ours	320	OpenVino	FP16	2.8	1.87
High-power	Xeon 4114	85W	Ours	608	OpenVino	FP16	5.3	0.06
	1080Ti	250W	Ours	320	MMDet	FP32	40.6	0.16
	1080Ti	250W	Ours	320	TensorRT	FP32	**186.6**	0.74
	1080Ti	250W	Ours	608	MMDet	FP32	32.8	0.13
	1080Ti	250W	Ours	608	TensorRT	FP32	68.2	0.27
	1080Ti	250W	Ours	416	MMDet	FP32	39.1	0.15
	1080Ti	250W	[Xu+19a]	416	PyTorch	FP32	38.5	0.15
	1080Ti	250W	[MBZ20]	800	MMDet	FP32	4.9	0.02
	1080Ti	250W	[MBZ20]	360	MMDet	FP32	16.5	0.07

Table 4.6: Throughput of SCR with Darknet-53 backbone on a wide range of devices with varying levels of power. H = image height.

4.4.9 Hardware configuration

This section contains details regarding the embedded hardware configuration used for the throughput benchmarks of the previous section.

Our Nvidia Jetson Nano 2GB, Nano 4GB and Xavier NX development kits are running JetPack 4.5, ONNXRuntime 1.4.0 and TensorRT 7.1.3. The Nano SoC features a quad-core ARM A53 CPU and a 128 CUDA core Maxwell GPU. In order to obtain consistent results, we lock the power consumption at 10W for benchmarking using the `jetson_clocks` tool and cool the heatsink with a Noctua A4x20 5V PWM fan using the default fan curve. We also ran our benchmarks on a Jetson Nano 4GB model and found no performance difference with the 2 GB model. Thus, Table 5.2 refers to both of them as "Nano". The Jetson Xavier NX features a 6-core ARM processor, a 384 CUDA core Volta GPU and 8GB RAM, with a power consumption of 30W and is running the same setup. We benchmark SCR on the Intel Movidius Myriad X using OpenVino and FP16 precision. Nvidia GeForce GTX 1080Ti (11GB VRAM, 250W) benchmarks were run using either the same MMDetection setup used for training, or TensorRT 7.1.3. Intel Xeon Silver 4114 CPU benchmarks were run using the Intel OpenVino toolkit.

4.5 Conclusion

In this chapter, we proposed SCR, a method that captures object contours with a compact, resolution-free representation based on a low number of Fourier coefficients. Using a complex representation and geometric priors, we observed qualitative and quantitative improvements in the regressed shapes compared to those produced by previous methods.

We also implemented a downsized version of our model, which, while retaining competitive accuracy, is able to run on a wide range of low-power hardware at very reasonable speeds (up to 115 FPS on a 30W device), and is, therefore, suitable for edge computing applications such as on-board processing of UAV or satellite images, autonomous vehicles, and robotics.

Future work will include testing and evaluating SCR on other applicative scenarios, such as autonomous driving and remote sensing. Matching the "shape signature" of detected objects to an on-board database is an example of how our work might be used for embedded change detection. This work could also be extended by adding a polygon simplification scheme that takes advantage of properties of the Fourier decomposition, such as ease of differentiation.

Chapter 5

Road graph extraction

In this chapter, we present a fast single-shot graph extraction architecture based on a Fully Convolutional Network and a Graph Neural Network, which we apply to road mapping.

5.1 Introduction

Automatic road graph extraction from aerial and satellite images is a long-standing challenge. Vector-based maps are heavily used in Geographic Information Systems (GIS) such as online maps and navigation systems. The extraction of roads from overhead images has been, and mainly still is, performed by expert annotators. For a long time, handcrafted methods have been developed with the aim of reducing the burden of the annotators. However, they were not precise enough to replace them. Indeed, their outputs were imperfect and had a high dependence on user-adjustable parameters [Lia+20].

Modern road extraction methods mostly fall into two kinds of approaches, each coming with its own drawbacks. The first kind [MLU17; ZLW18; ZZW18; Bat+19] relies on a pixel-based segmentation which is then converted to a graph using compute-intensive handcrafted algorithms that require a lot of manual tuning and can often yield partial graphs. The production of raster road masks using neural networks is a computationally expensive task in itself because of the large number of layers required to retrieve a segmentation that has the same resolution as the input image. Iterative graph construction methods are the second kind of deep learning based methods for road extraction [Bas+18; LWL19; Tan+20; LH20]. These approaches explore the map from an initial starting location using successive applications of a neural network on patches centered on the current point in order to predict the next-move. While having the advantage of directly inferring a graph, this is a very slow process. In addition, several starting locations may need to be

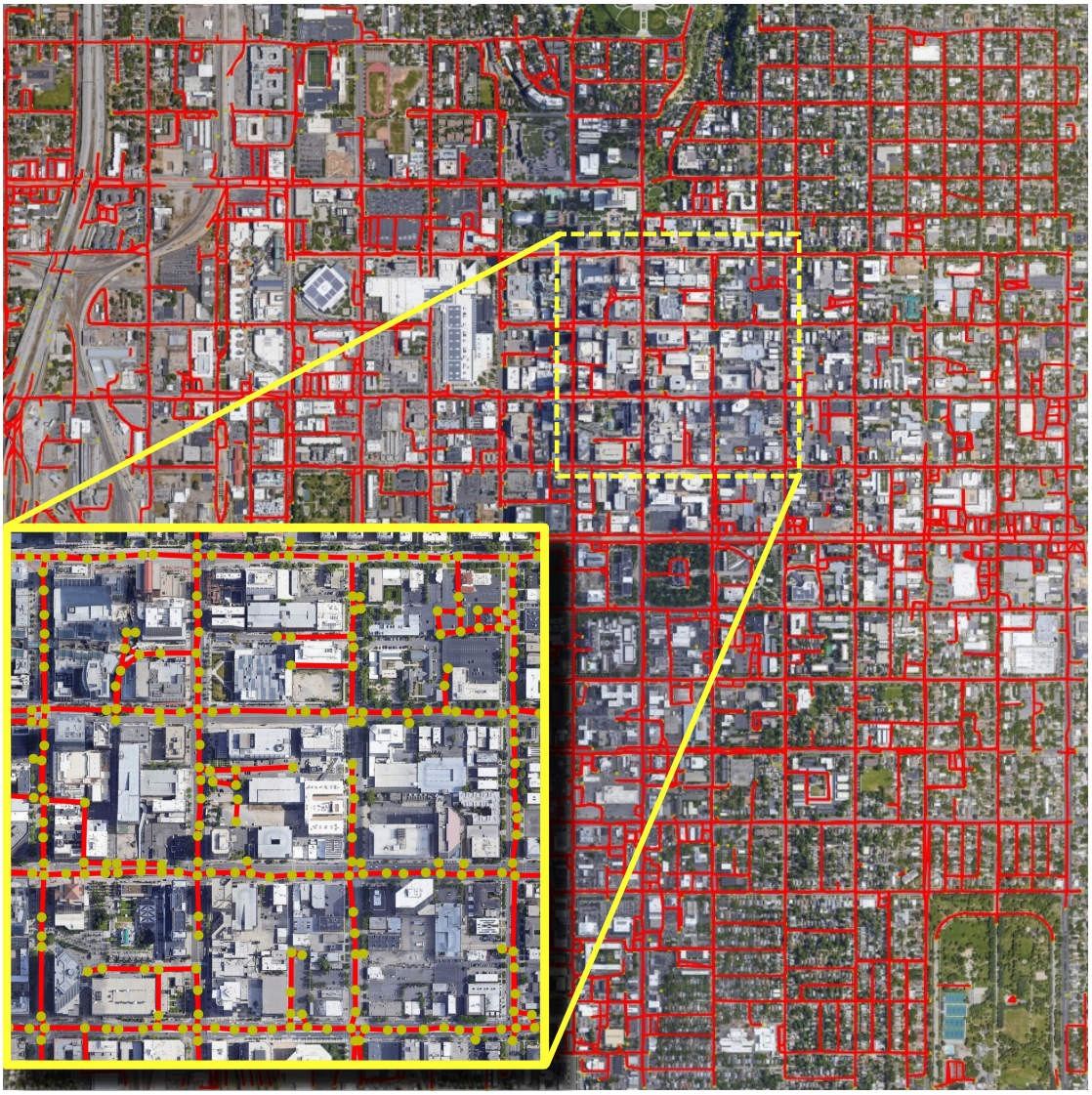


Figure 5.1: Result of our method on an image from the RoadTracer [Bas+18] test set. Our method regresses an entire graph without the need of pre or post-processing. This 8192x8192 pixel image was processed in 22 seconds on a single GPU, which is up to **91 times faster** than previous approaches. The extracted graphs have a low complexity and retain good accuracy compared to existing works.

picked in order to cover the possible multiple connected components of the final graph.

Our approach aims to solve the aforementioned issues of current deep learning based methods by offering a fast and convenient way to extract the final road graph from satellite or aerial images in a single pass. Inspired by recent advances in neural networks for object detection [Tia+19], we design a fully convolutional detection head that learns to precisely locate multiple points of interest such as intersections, turns, and dead ends across the whole image in a single pass. Node features attached to these points are simultaneously regressed by this head, and fed to a Graph Neural Network (GNN) which predicts links between these points to form the final graph without requiring any post-processing. Both networks are jointly trained end-to-end from ground truth road graphs, which are widely available from sources such as OpenStreetMap. Such a strategy is more efficient than iterative methods and allows us to streamline the training process by removing the need for generation of starting locations. We benchmark our method against state-of-the-art approaches on the RoadTracer [Bas+18] dataset and find that it is orders of magnitude faster than recent competing methods while retaining competitive accuracy. With the increasing interest in deep learning inference on embedded hardware, such as satellites or drones [Bah+19], our method, combined with recent advances in CNN and GNN quantization [BT19; BBZ21], opens the possibility of on board in-flight road extraction, which could reduce ground computation and bandwidth needs by transmitting graphs instead of images.

The main contribution of this work is threefold:

1. We propose a fast fully convolutional method for precise localisation of points of interest.
2. We design a Graph Neural Network for road link prediction.
3. We propose a performance-oriented graph extraction architecture highly competitive against state-of-the art approaches.

5.2 Related Works

5.2.1 Convolutional Neural Networks

In recent works, Convolutional Neural Networks (CNN) such as VGG [SZ14] or ResNet [He+16] which were originally used for image classification, are stripped of their final classifier and repurposed as fully convolutional feature extractors (often called backbones) for other computer vision tasks, such as semantic segmentation [LSD15]. Object detection methods such as YOLO [Red+16] or FCOS [Tia+19]

have taken advantage of the fully convolutional nature of these backbones to design very fast single stage object detection neural networks, which infer bounding boxes and classes in a single forward pass and can be trained end-to-end, as opposed to two-stage approaches that perform these tasks separately. Single-stage architectures often follow a similar structure composed of a backbone which extracts features, a neck outputting feature representations at different scales [Lin+17a], and several detection heads performing the final task.

5.2.2 Road Extraction

A large number of unsupervised methods have been developed for road graphs extraction. We refer the reader to recent extensive surveys on the subject for more information [LH20]. In this section, we focus on recent road extraction methods based on deep learning. Early deep learning based methods for road extraction were patch-based applications of fully connected neural networks [MH10]. However, most recent neural network-based methods proceed differently and can be sorted into two groups.

The first group of methods is based on pixel-level segmentation. Inspired by the success of FCN [LSD15] and U-Net [RFB15], these methods use a backbone as an encoder and learned upscaling as a decoder to retrieve a classification of individual pixels of an input image. In [ZLW18], residual connections are added to a U-Net to improve road predictions. D-LinkNet [ZZW18] adapts the LinkNet architecture to form a U-Net-like network and achieve better connectivity. DeepRoadMapper [MLU17] use a ResNet-based[He+16] FCN to produce a segmentation which is then converted to a graph using thinning and an additional connection classifier. In [Bat+19], the authors improve the connectivity of extracted networks by jointly learning the segmentation and orientation of roads. All segmentation-based approaches have the drawback of relying on post-processing techniques to convert the predicted road pixels to a road graph. Thus, these methods are slow and the final graphs can lack connectivity.

The second kind of approaches iteratively construct a graph from an initial starting point by successive applications of a neural network for next move prediction around the center of an input patch. These methods are inspired by the way human annotators iteratively create road graphs. In [Ven+18], a network predicts which points from the border of the output patch are linked to the center point. RoadTracer [Bas+18] uses a CNN to predict the direction of the next move with a fixed step size. PolyMapper [LWL19] uses a Recurrent Neural Network (RNN) guided by segmentation cues to predict the road topology and building polygons. VecRoad [Tan+20] implements a variable step size for next move prediction in order to achieve better alignment of intersections. DeepWindow [LH20] estimates the road direction along a center line regressed by a CNN using a Fourier spectrum

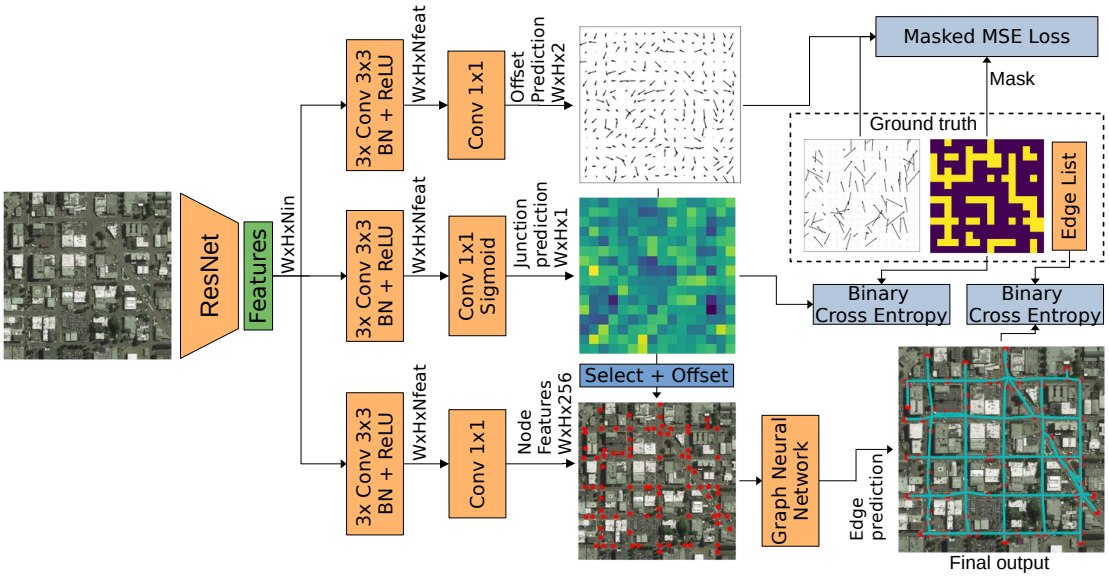


Figure 5.2: Our single-shot road graph extraction neural network. Features computed using a ResNet [He+16] backbone are fed to three branches. The junction prediction branch identifies cells which contain points of interest and is trained with binary cross-entropy. The offset prediction branch regresses the precise location of these points inside of the cells by outputting a two-dimensional vector field and is trained with a Mean Squared Error loss. Points selected by the junction and offset branch are paired with node features from a third branch and fed to a Graph Neural Network which predicts the presence of edges between pairs of points to create the final output.

analysis algorithm. While these methods have the advantage of outputting a road graph directly, they cannot guarantee the exploration of the whole map from a single starting location. The choice of initial starting locations is a hard problem in itself. Moreover, the repeated application of CNNs makes these approaches very slow.

5.2.3 Relational Inference and GNNs

Graph Neural Networks (GNNs) [Sca+09] are effective models for encoding interactions, and have recently been applied to link prediction [ZC18], relational structure discovery, such as to model the dynamics of complex systems [Kip+18], and to the modeling of relational knowledge [Sch+18]. Kipf and Welling [KW16] proposed a graph autoencoder using a Graph Convolutional Network (GCN) encoder [KW17] and a simple dot-product decoder to model undirected networks and applied it to the link prediction task. Follow-up work [Sch+18] models different edge types in relational data using a graph neural network encoder and a decoder

based on the DistMul [Yan+14] factorization. In [Kip+18] the authors propose a message-passing [Gil+17] encoder that alternates between the computation of node and edge features that can later be leveraged to predict the evolution of the system of interest several time steps in advance. The aforementioned models assume the ground-truth graph is partially known, or operate on the complete graph as an uninformative prior [Kip+18]. A parallel line of work is that of learning the unknown graph explicitly. The Dynamic Graph CNN model [Wan+19b] proposes to dynamically build a graph by k-NN search in the feature space after each application of the EdgeConv operator also introduced in [Wan+19b]. The approach has been extended in [Kaz+20], where the authors address the question of the differentiable construction of a discrete graph using the recently proposed Gumbel Top-k trick [KVW19] - a stochastic and differentiable relaxation of k-NN search - and decouple the construction of the graph and the learning of graph features amenable for downstream tasks. With the introduction of deeper architectures [Li+19a; Gon+20] came increased interest for efficient implementations of GNNs, an issue relevant to our work as on-board processing of satellite images, such as for road graph extraction, is a pressing issue. [Wan+20a] applied bucket-based quantization of matrix-matrix products to accelerate the GCN [KW17] operator. [BBZ21] proposed a general framework for binarizing graph neural networks and specifically introduced efficient binarized versions of the Dynamic EdgeConv operator [Wan+19b] with real-world speed-ups on a low-power device.

5.3 Method

In this section, we present our method based on a fully convolutional head for regression of points of interest and a Graph Neural Network (GNN) for link prediction. An overview of our method is available in Figure 5.2.

5.3.1 General architecture

Our neural network architecture follows the trend of using a pre-trained CNN as a feature extractor (backbone). Most detection neural networks also use a Feature Pyramid Network (FPN) as a "neck" that aggregates features from multiple scales. However, we chose not to use an FPN since satellite images have a fixed ground sampling distance and consequently, models that work on them are less vulnerable to changes in scale. Since we work at a single level of detail, we feed the output feature maps of the feature extractor directly to a single head. These changes allow us to save computation time and make the training process more streamlined. Indeed, we remove all ground truth bounding box assignment complications that arise when using multiple heads.

Each branch of our head is composed of three convolutional layers working on N_{feat} feature maps. The first two branches are the junction-ness and offset branch, which form the point-of-interest detection model. The third branch is a node feature branch, which computes features that will be used to predict links between points detected by the "point-of-interest" branch.

Let H_I, W_I be the dimensions of the input RGB image $I \in \mathbb{R}^{H_I \times W_I \times 3}$. With the ResNet-50 [He+16] backbone used in our experiments, the height and width H, W of the input feature maps $F^{in} \in \mathbb{R}^{H \times W \times N_{in}}$ are 32 times smaller than the ones of I. This means that the final layers of our head have $H \times W$ output cells. In the case of an input image size of 512×512 pixels, for example, we thus obtain 256 output cells, which can each regress the position of one point.

5.3.2 Point detection branch

Our point of interest detection branch, shown in Figure 5.2 is inspired by recent developments in single shot detection neural networks [Tia+19; Red+16], which regress a bounding box for each output "cell" of a detection head, even if they do not necessarily contain an object. This allows object detection to be performed using very fast Fully Convolutional Networks. In [Tia+19], the notion of "center-ness" is then introduced to filter out cells that do not belong to any object. These methods represent bounding boxes as four offsets relative to the center of the cell. We adapt this design to the regression of points of interest by outputting a "junction-ness" score J_j for each output cell j, as well as a two-dimensional vector $\mathbf{o}_j = [u_j, v_j] \in [-0.5; 0.5]^2$ representing the offset of the regressed point with respect to the center of its cell. A point $\mathbf{p}_j$ is detected in an output cell j if $J_j > J_{thr}$, where J_{thr} is the junction-ness detection threshold. We can retrieve the coordinates (x_j, y_j) of $\mathbf{p}_j$ in the original input image:

$$x_j = \frac{u_j + X_j + 0.5}{W} \cdot W_I, y_j = \frac{v_j + Y_j + 0.5}{H} \cdot H_I \tag{5.1}$$

where (X_j, Y_j) are the coordinates of the cell j in the output feature map F^{out}.

5.3.3 Choice of input resolution

By design, our method is only able to find a single point of interest by output cell and is directly trained on ground truth annotations. However, depending on the dataset, and especially in very dense neighborhoods, several ground truth points may fall into the same cell. We chose that these points would effectively be merged into their centroid, while keeping incoming edges from outside of that cell linked to that new point, as shown on Figure 5.3. However, this situation should be avoided as much as possible, since it can shift intersections, and link close points that

should not be linked (e.g. parallel roads). Images are often resized when being forwarded through a neural network, as a compromise between speed, memory and accuracy. In our case, the resizing ratio has to be chosen carefully according to the ground sampling distance (GSD) of the dataset and the density of ground truth annotations. To estimate the correct ratio for each dataset, we evaluate the average number of points in each positive cell over the whole dataset, at a wide range of ratios. This average should be as close to one as possible, in order to limit the effects of point merging. Figure 5.4 shows this average for the popular RoadTracer [Bas+18], SpaceNet3 [VLB18] and Massachussetts roads [Mni13] datasets.

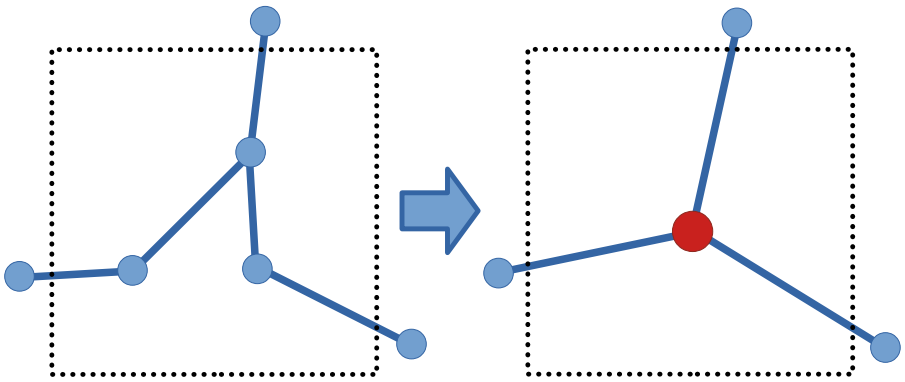


Figure 5.3: Node averaging which occurs when several ground truth points belong to the same output cell. The offset branch is thus trained to regress the centroid of these ground truth points.

5.3.4 Edge prediction

Once junctions have been extracted from the image, our task is to predict which ones are connected. We cast the problem as that of learning a latent graph. Starting with an initial prior connectivity estimation - *i.e.* a set of edges - $\mathcal{E}^0$, we aim at both inferring missing edges and discarding irrelevant ones.

Formally, let j be one of the detected junctions. We denote by $X_j = F_j^{out}$ the corresponding features in the output feature map of the backbone, and $\mathbf{p}_j = (x_j, y_j)$ the 2D Cartesian coordinates of the junction in the input image, computed from the offset vectors.

Our method computes initial node embeddings

$$\mathbf{x}_j = f(X_j, \mathbf{p}_j). \tag{5.2}$$

We choose the function f to be either a 2D Convolutional Neural Network defining an additional node features branch of the network - responsible for dimensionality

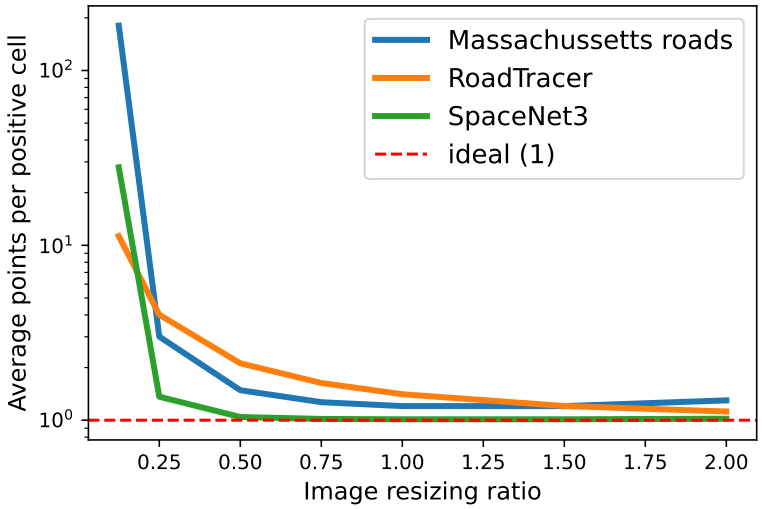


Figure 5.4: Average ground truth points per positive output cell at a range of image resizing ratios for 3 popular road extraction datasets. The ideal average is one, to prevent the collapse of neighboring nodes as much as possible.

reduction and transformation of the raw image features - or the identity, the latter case leaving the task of transforming the raw image features to the GNN. We then perform several (*e.g.* three) message-passing iterations using the EdgeConv operator [Wan+19b]:

$$\mathbf{e}_{ij}^{(l)} = \text{ReLU}\left(\boldsymbol{\theta}^{(l)}(\mathbf{x}_j^{(l-1)} - \mathbf{x}_i^{(l-1)}) + \boldsymbol{\phi}^{(l)}\mathbf{x}_i^{(l-1)}\right) \tag{5.3}$$

$$= \text{ReLU}\left(\boldsymbol{\Theta}^{(l)}\tilde{\mathbf{X}}^{(l-1)}\right) \tag{5.4}$$

$$\mathbf{x}_i^{(l)} = \max_{j \in \mathcal{N}(i)} \mathbf{e}_{ij}^{(l)} \tag{5.5}$$

followed by Batch Normalization, where $\tilde{\mathbf{X}}^{(l-1)} = \left[\mathbf{x}_i^{(l-1)} \,||\, \mathbf{x}_j^{(l-1)} - \mathbf{x}_i^{(l-1)}\right]$, $\boldsymbol{\theta}$ and $\boldsymbol{\phi}$ are trainable weights, and $\boldsymbol{\Theta}$ their concatenation. The resulting increase in receptive field allows for connectivity patterns to be learned based on both local image and graph properties, and shared context across junctions.

Finally, each possible edge is scored using a simple scoring function g applied on the final node features of the graph. While we could opt to classify the edges based on the computed edge features $\mathbf{e}_{ij}$, performing message passing on a sparse graph (potentially built dynamically) is desirable for efficiency at both training and inference. We note

$$p_{ij} = \sigma(g(\mathbf{x_i}, \mathbf{x_j})) \tag{5.6}$$

the inferred probability that the edge $\mathbf{e}_{ij}$ exists, where σ is the sigmoid function.

In practice, we choose g to be either a single bilinear layer, *i.e.*

$$g(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T W_g \mathbf{x}_j \tag{5.7}$$

or a multi-layer MLP classifier (please refer to our ablation study in section 5.4.10 for more details and comparison with other decoders such as the dot product).

5.3.5 Connectivity estimation

Our choice of operator is motivated by the fact that the connectivity is unknown, and therefore we may choose $\mathcal{E}^0$ to be the set of edges of the complete graph (*i.e.* considering all possible connections between junctions), or introduce a sparse prior by building the graph dynamically by k-NN search in feature space. We also considered the case where the initial graph is complete, while subsequent iterations of message passing are done on dynamically inferred connectivity. Choosing $k = 4$ is motivated by the observations that junctions with more than four incident roads are rare. To verify this intuition, we trained a model using a three-layer (2D convolutions followed by Batch Normalization and ReLU activations) node features head and an MLP classifier for each possible edge. The model was trained to convergence, and the four nearest neighbours graph was built on the output of the node features branch. The resulting graph is shown in Figure 5.5, and demonstrates that a high quality sparse initialization of the connectivity can be obtained in this manner.

5.3.6 Loss function

To train our neural network our loss function is defined as a sum of three terms:

$$\mathcal{L} = \mathcal{L}_{\text{jun}} + \mathcal{L}_{\text{off}} + \mathcal{L}_{\text{edge}} \tag{5.8}$$

where $\mathcal{L}_{\text{jun}}$ and $\mathcal{L}_{\text{edge}}$ are the binary cross-entropy loss which we use to train our junction-ness branch and our edge prediction Graph Neural Network.

$\mathcal{L}_{\text{off}}$ is a Mean Squared Error offset loss masked with the positive junction predictions, which we use to train our offset regression branch. We define it as:

$$\mathcal{L}_{\text{off}} = \frac{1}{\sum \mathbb{1}_{J_{i,j} > J_{thr}}} \sum_{i,j} \mathbb{1}_{J_{i,j} > J_{thr}} (v_{i,j} - V_{i,j})^2 \tag{5.9}$$

Where V is the ground truth offset vector field. The mask is used so that the predictions of the offset branch are not conditionned to the presence of nodes.

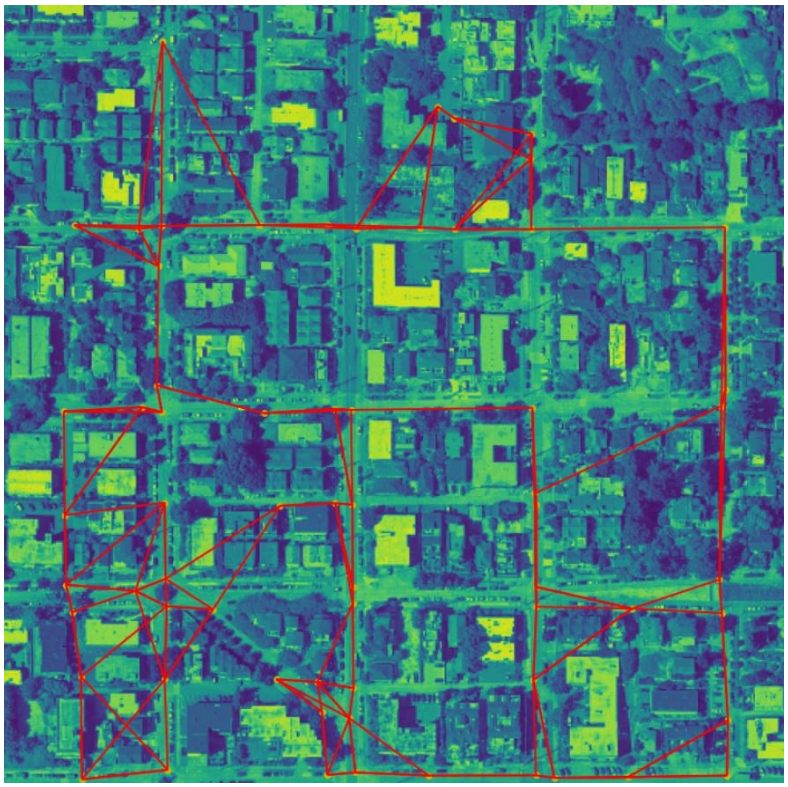


Figure 5.5: $k = 4$-NN graph constructed in feature space on features learned by an additional *node features head* combined with an MLP edge classifier.

5.4 Experiments

In this section, we compare our method against other road extraction approaches on the widely-used RoadTracer dataset [Bas+18], which is composed of 35 training cities (180 images of 4096x4096 pixels) and 15 test cities of 8192x8192 pixels. The low number of training images, low ground sampling distance, high resolution and high annotation density of this dataset make it quite challenging.

5.4.1 Implementation details

According to the results shown in Figure 5.4, we choose a rescaling ratio of 0.5 as a good compromise between accuracy and speed. We implement our method in the Pytorch 1.9.1 [Pas+19] framework based on CUDA 11. We choose a ResNet-50 backbone (thus $N_{in} = 2048$) and $N_{feat} = 256$. We perform edge classification on the complete graph (other cases included in the ablation study 5.4.10) using a three-layer GNN and a 2-layer MLP scoring function.

We use basic data augmentation techniques such as random flips, and train our network on 512x512 pixel random crops for 2350 epochs, which took 24 hours on our system with a single Nvidia RTX 3090 GPU with 24GB of VRAM, an AMD Ryzen 9 3900X CPU and 64GB of RAM. We use the Adam [KB14] optimizer with a learning rate of $1e^{-3}$.

We also run performance benchmarks. All numbers were obtained on a computer with two Intel Cascade Lake 6248 20-core CPUs, 192GB of RAM, and four Nvidia Tesla V100 GPU with 16GB of VRAM. 10 CPU cores, a single GPU, and a quarter of the available memory were assigned to each run.

5.4.2 Inference on large images

In our testing, we found that the output of the link prediction branch is tied to the original training resolution of the network. Thus, we did not obtain good results by directly inferring on a bigger image. Instead, we apply our network on large images using a sliding window of the same size as the training resolution, while averaging the node positions and accumulating the detected edges.

5.4.3 Qualitative results

Figure 5.6 shows qualitative results against an iterative method and a segmentation-based method. Our method is able to find a noticeably larger number of roads than RoadTracer [Bas+18] and DeepRoadMapper [MLU17] in this challenging image. We can also notice that there are a few areas with low connectivity in our result compared to RoadTracer. Indeed, since their network "walks" through the entire

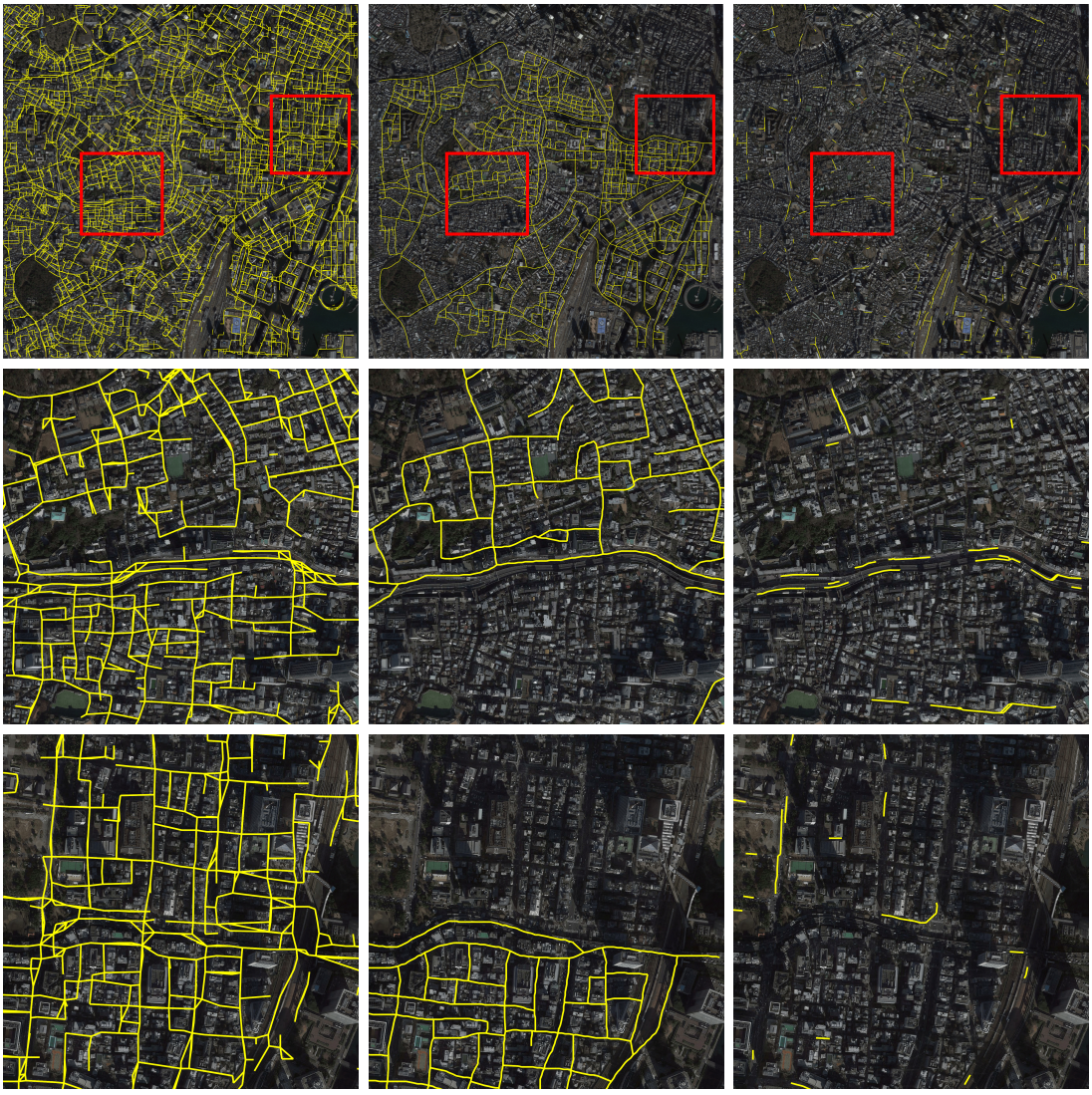


Figure 5.6: Comparison of our method against an iterative method [Bas+18] and a segmentation based method [MLU17] on a challenging area of the RoadTracer dataset. The first line is the full image. The second and third line are crops of the regions in red squares. Our method is able to find more roads, even in sections that are completely missed by other methods.

Method	P-F1	J-F1	APLS
DeepRoadMapper [MLU17]	56.85	29.05	21.27
RoadTracer [Bas+18]	55.81	49.57	45.09
RoadCNN[Bas+18]	71.76	–	–
VecRoad[Tan+20]	**72.56**	**63.13**	**64.59**
Ours	57.2	39.23	46.93

Table 5.1: Accuracy metrics on the RoadTracer Dataset, evaluated using open source code from [Tan+20] and [Bas+18]. RoadCNN is a purely segmentation-based method implemented by [Bas+18], thus it only has a P-F1 score.

regressed graph, it is inherently made of a single connected component. However, the result of RoadTracer is missing entire portions of the city (e.g. top left) because they are separated from the starting position by a highway that the network was not able to cross during its exploration. Our method does not suffer from such limitations.

Figures 5.1, 5.12 and 5.13 show more qualitative results on cities from the RoadTracer test set. Our method is able to find more roads than RoadTracer [Bas+18] and DeepRoadMapper [MLU17]. Some roads and highways that cannot be accessed easily by iterative approaches are found by our method. Our graphs also seem to have a better connectivity than the ones found by the segmentation-based method.

5.4.4 Quantitative evaluation

We follow the literature and use the three metrics defined in [Tan+20] to evaluate the accuracy of our network. P-F1 is a pixel-based F1 score obtained by comparing the rasterized output graph and ground truth. J-F1 is a junction-based F1 score based on local connectivity. APLS is the Average Path Length Similarity defined in the SpaceNet challenge [VLB18] and is based on the comparison of shortest paths in the predicted graph and the ground truth.

The results of our testing are shown in Table 5.1. Our method achieves results which are competitive with DeepRoadMapper [MLU17] and RoadTracer [Bas+18]. It is however outperformed by VecRoad [Tan+20]. Our intuition is that VecRoad sort of "brute-forces" these metrics by regressing a lot more points and edges than our method, as shown in Section 5.4.6 and Table 5.3.

5.4.5 Performance benchmarks

Run-time is seldom mentioned in the road extraction literature, but it is a very important metric, especially as we move towards inference on edges devices. Thus,

Method	Type	Single image	Whole dataset
VecRoad [Tan+20]	Iterative	1902.5	8873.4
RoadTracer [Bas+18]	Iterative	579.4	8690.5*
DeepRoadMapper [MLU17]	Seg.	1361.6	20423.4
RoadCNN [Bas+18]	Seg.	58.6	878.5
Ours	Graph	**20.9**	**295.0**

Table 5.2: Run-time in seconds on the RoadTracer Dataset. The single image score is averaged over the whole test set. Our method is the fastest by a large margin. *RoadTracer failed on some images, thus this number is extrapolated from the average.

we benchmark the performance of our method against other recent approaches with open source code. We remove data loading and output times from all methods and thus only account for pre-processing, inference and post-processing steps of each algorithm. We provide average performance numbers for a single 8192x8192 test image from the RoadTracer [Bas+18] dataset, as well as numbers for the whole dataset (15 images), since some methods such as VecRoad [Tan+20] have optimizations for multiple-image inference. Our method can run on the whole dataset at once by performing the sliding window in batches, which is a bit faster.

The results of this experiment are shown in Table 5.2 and Figure 5.7. We observe that iterative methods are much slower because of the repeated forward passes required for next move estimation. Segmentation methods are not particularly fast as well, as the upscaling computation performed by a learned decoder is very compute intensive, and the extra post-processing steps required for graph conversion are slow. Our method is the fastest one by a large margin and offers a very good compromise between accuracy and speed. Indeed, it achieves similar APLS scores as RoadTracer [Bas+18] while running 27 times faster. Our approach also beats the APLS scores of DeepRoadMapper [MLU17] by a large margin while running 65 times faster. It has lower accuracy scores than VecRoad [Tan+20], however it is 91 times faster. Our method could be made even faster by using a smaller and more efficient backbone, such as ResNet-18 [He+16] or DarkNet-53 [Red+16], as shown in Section 5.4.7.

5.4.6 Graph complexity

Another interesting metric which is seldom mentioned in the road extraction literature is the notion of graph complexity. Graphs are a sparse representation and are meant to use a lot less storage space than masks. Most algorithms commonly used on graphs (e.g. shortest path algorithms) have a complexity which depends on the

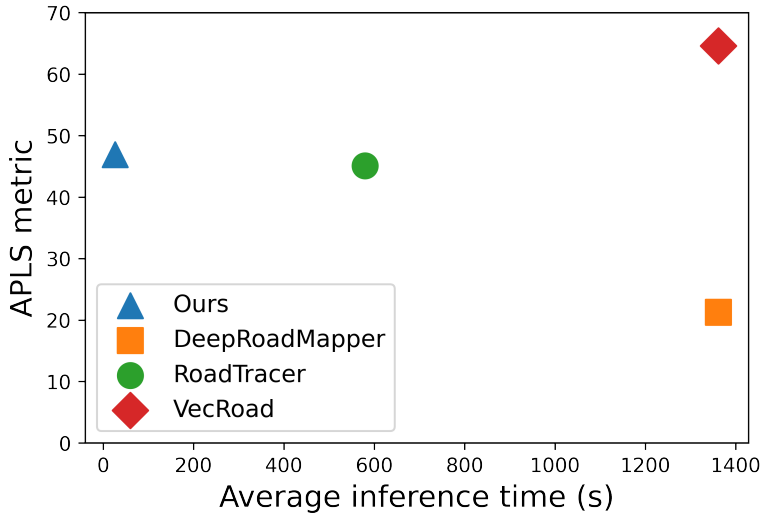


Figure 5.7: APLS metric with respect to inference time for our method, compared to state-of-the art approaches. Our method is the fastest and offers a very good compromise between accuracy and speed.

number of nodes and edges of the graph. Thus, offering accurate graphs with a low number of unnecessary nodes and edges is also beneficial in terms of run-time of subsequent applications. To evaluate the complexity of the graphs regressed by our method and compare it to other approaches, we propose a complexity score which is simply the total number of graph elements (nodes and edges) divided by the APLS score (lower is better). We use the APLS because it seems to be the most popular metric. This score represents the compromise between the accuracy and compactness of graphs.

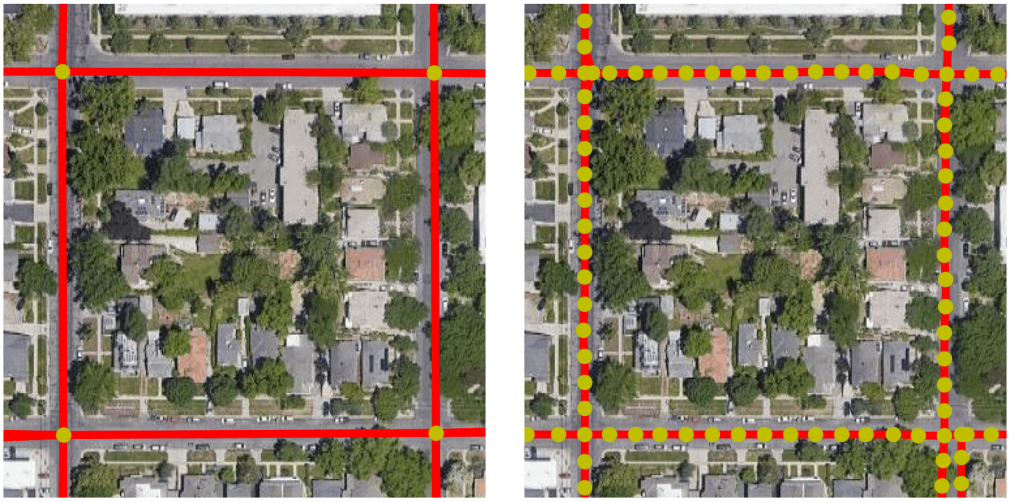
The results of this experiment are shown in Table 5.3. We observe that our method obtains the lower complexity compared to VecRoad [Tan+20] and DeepRoadMapper [MLU17]. This is mainly due to our sparse approach to the regression of junctions, as shown by the average number of nodes. In Figure 5.8, we show the way our method finds an optimal representation of a neighborhood compared to the very high number of nodes and edges found by VecRoad [Tan+20].

5.4.7 Impact of choosing a smaller backbone

Feature extractors (backbones) come in different sizes to suit different kinds of performance targets. For example, some applications like edge computing might accept a reduction on accuracy in favor of faster inference time and higher throughput. In order to evaluate the impact of choosing a smaller backbone on our method, we take the network presented above and replace the ResNet-50 backbone with

Method	Nodes	Edges	Total	APLS	Complexity
VecRoad [Tan+20]	29620	61042	90662	**64.59**	1404
DRM [MLU17]	6071	11963	18034	21.27	848
RoadTracer [Bas+18]	8263	17044	25306	45.09	561
Ours	4343	17273	21615	46.93	**461**

Table 5.3: Comparison of average graph complexity scores (Total elements divided by APLS. Lower is better). Nodes (resp. Edges) is the average number of nodes (resp. edges) in the output graphs over the whole RoadTracer test set. DRM = DeepRoadMapper. Our method achieves the lowest complexity compared to other approaches, which make it a good compromise between accuracy and compactness of graphs.



Ours VecRoad [Tan+20]

Figure 5.8: Exemple of the low complexity of the graphs inferred by **our method (left)** compared to VecRoad [Tan+20] (right). Our method finds an optimal 4-node representation of this simple neighborhood, whereas iterative methods tend to find overly complex representations. Having graphs with a low number of elements will save storage and speed up the computation of algorithms performed on these graphs, such as shortest path algorithms.

Method	APLS	Single image time (s)
Ours (R18)	38.80	**19.6**
Ours (R50)	**40.45**	20.9

Table 5.4: Run-time in seconds on the RoadTracer Dataset. The single image score is averaged over the whole test set. Using a smaller backbone (R18) is slightly faster but causes a small drop in APLS.

ResNet-18.

The results of this experiment are presented in Table 5.4. We observe that using a ResNet-18 backbone is slightly faster but results in a slightly lower APLS score. This means that our method works with smaller backbones and thus can be used on a variety of low power devices, especially ones that have a low amount of memory, such as the Nvidia Jetson family of devices.

5.4.8 Impact of edge detection threshold

In this section, we study the impact of the edge detection threshold, which can be freely chosen between 0 and 1. To do this, we compute the three main accuracy metrics for a range of thresholds varying between 0 and 0.5, using our main model. Setting a high detection threshold favors the Precision of edges, while setting a low threshold favors the Recall. Figure 5.9 shows the results of this experiment. We observe that the P-F1 (pixel metric) and APLS are better when detecting more edges, while the J-F1 (junction metric) is slightly better when detecting fewer edges. This makes sense as the J-F1 is based on having the correct number of edges for each junction, while the APLS should benefit from more path options. The P-F1 may be higher at low thresholds simply because we find more road pixels. This experiment shows that finding and setting an appropriate edge threshold for each model is important in order to obtain the best accuracy and the best compromise between the three metrics.

5.4.9 On the J-F1 metric

As previously said in sections 5.3 and 5.5, our method is only able to find a single point-of-interest per output cell, which can lead to merged junctions in the output graph. In addition, as seen in section 5.4, our method will inherently find a lower number of junctions since it is tailored towards the extraction of a sparse graph. Since the J-F1 score is based on a matching of junctions within a certain radius, we believe that these design choices significantly affect this metric, which leads to our method scoring lower than some other approaches.

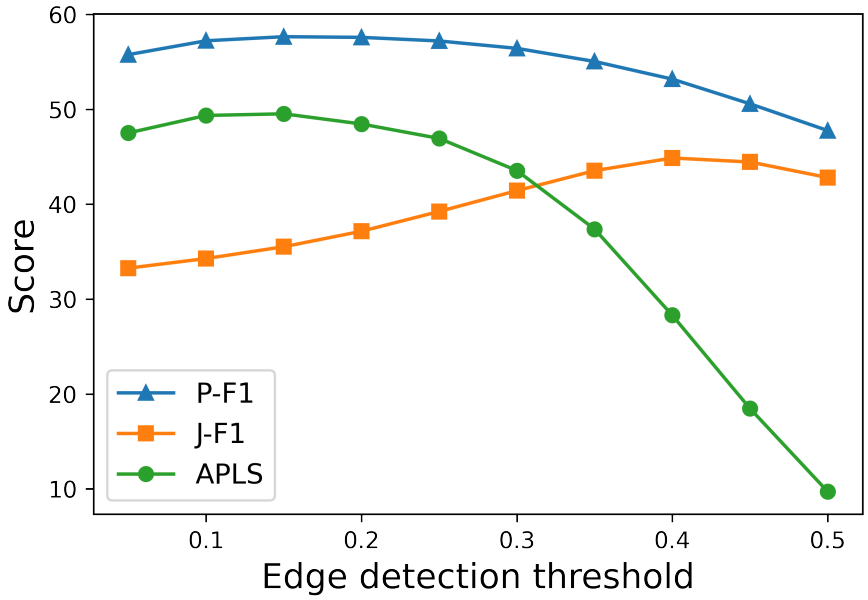


Figure 5.9: J-F1, P-F1 and APLS scores for our main model, at a range of edge detection thresholds.

5.4.10 GNN model choices and ablation study

In this section, we perform a comparative study of different design decisions for the GNN portion of the model.

We compare the following choices for the construction of the road graph:

1. Using the complete graph as the supporting graph for each GNN layer and treating the problem as a pure edge classification task
2. Initializing the road graph by k-NN search (k = 4) on the output of the node feature branch (dim = 256) to which we concatenate the Cartesian coordinates of the junctions (dim = 2)
3. Dynamic construction of the graph at each layer by k-NN search (k = 4) on the layer's input (*i.e.*, as described in point 2 for the first layer, and on the output features of the previous layer for subsequent layers)

We also compare two choices of edge scoring function:

1. A 2-layer MLP (FC(256) $\rightarrow$ FC(256) where FC(N) denotes a fully-connected layer with N output features)
2. A $256 \times 256 \times 1$ bilinear layer

We use a three-layer graph neural network based on the EdgeConv operator with BatchNormalization and ReLU activations. We parameterize the EdgeConv operators each with a linear layer with 256 output features. We also report the results of baseline models trained without graph convolutions.

Finally, we compare the 3-layer EdgeConv networks with DeepGCNs [Li+19b; Li+20] using the Generalized Graph Convolution (GENConv) operator [Li+20], layer normalization [BKH16] and ReLU activations. We apply the DeepGCN layers sequentially on:

1. The raw image features produced by the backbone (DeepGCN)
2. The output of one convolutional layer in the node feature branch (1 Conv + DeepGCN)
3. The output of 4 convolutional layers in the node feature branch (4 Conv + DeepGCN) as for the EdgeConv-based models

In case (1) we used 7 layers of (GENConv $\rightarrow$ LayerNorm $\rightarrow$ ReLU), in case (2) we used 6 layers, and in case (3) we used 3 layers, so as to keep model capacity comparable with the EdgeConv models applied on the full node feature branch. We applied the DeepGCNs on the complete graphs only.

For each of these experiments, we report the J-F1, P-F1 and APLS metrics at the edge threshold that maximizes their sum. Table 5.5 shows the results of these experiments. To enable faster experimentation, we initialized some of the models using the pre-trained weights of a "Baseline MLP" model trained on the complete graph, such models are denoted by a checkmark in the "PT" (*i.e.* "Pre-trained") column of Table 5.5.

We can draw the following conclusions from the experimental results shown:

- The baseline with the bilinear classifier outperforms the baseline with a 2-layer MLP, which is expected since it has a much larger number of parameters. While the improvement is noticeable, the model loses compactness and efficiency.
- All three constructions of the graph (complete, static k-NN and fully dynamic) are able to perform well and to outperform the MLP and the Bilinear baseline. Notably, the models that combine a GNN with an MLP classifier outperformed the ones with the same GNNs but a bilinear classifier (scoring function). The entire GNN adds fewer parameter than changing the MLP for a bilinear layer, and yet can bring larger performance deltas, which further motivates our choice of using an MLP scoring function.

- The best performing GNN with the EdgeConv operator used a fixed 4-NN graph support for message passing (but still scored all possible edges for the final graph). However, choosing the right value of k adds another element to hyperparameter tuning of the model and comes at the disadvantage of reduced robustness to cases where images have no junctions to detect (*e.g.* satellite images of large bodies of water such as lakes). We therefore chose to report the performance of the simpler model in section 5.4, while showcasing that sparse graph priors may indeed lead to better road graph reconstructions.

- Getting rid of the node features branch reduces the number of trainable parameters but appears to be detrimental to the performance in most cases. Experiments with deeper GNNs applied directly on the output of the backbone showed increased performance compared to shallower GNNs, although the graph construction differs (l. 14, 17)

- Compared to 3-layer EdgeConv networks on the complete graphs, 3-layer DeepGCNs applied following either a 1-layer or 4-layer node feature branch performed better. These models also outperformed the dynamic graph models and the 3-layer EdgeConv applied on a sparse 4-NN graph (by a slimmer margin in the latter case). The DeepGCNs applied on raw features outperformed some of the shallower models trained with a node feature branch, but did not match the best performing models.

- All DeepGCNs outperformed the MLP baseline, which indicates that, keeping the edge scoring function the same, replacing all (DeepGCN, l.17) or most (1 Conv + DeepGCN, l.18) 2D convolutions with graph convolutions leads to increased performance compared to a model that only uses 2D convolutions.

Deep Graph Convolutional Networks

We suspect the last two points are due to two effects: first, 2D convolutions can be seen as special cases of graph convolutions applied to the 2D lattice graph while enforcing translation equivariance; their inductive bias is well suited to the processing of images. In contrast, we applied the GNNs (EdgeConv or DeepGCN) on the (dynamic or complete) graphs of detected junctions, which means the GNNs do not have access to the neighboring pixel's context whereas the 2D Euclidean convolutions do. We believe this contributes to the reduced performance of all graph-convolutional models compared to models that combine 2D convolutions and graph convolutions. Additionally, the higher performance of the graph convolution models compared to only using 2D convolutions - especially on the APLS metric - show the contribution of the graph-based models.

#	PT	Init Graph	GNN	Classifier	RF	Wait	j_{thr}	J-F1	P-F1	APLS	Sum
1	✓	Baseline	✗	Bilinear				**40.33**	55.93	43.95	140.21
2		Baseline	✗	Bilinear				36.03	53.87	41.64	131.54
3		Baseline	✗	Bilin small				32.24	50.4	36.11	118.75
4	✓	Baseline	✗	MLP				36.92	53.43	41.17	131.53
5		Baseline	✗	MLP				35.46	56.03	45.35	136.84
6	✓	Complete	EdgeConv	Bilinear				37.69	56.04	45.39	139.13
7*	✓	Complete	EdgeConv	MLP				39.23	57.2	46.93	143.36
8	✓	Dynamic	EdgeConv	Bilinear				35.63	56.29	46.04	137.96
9	✓	Dynamic	EdgeConv	Bilinear			0.3	37.90	55.5	43.68	137.08
10	✓	Dynamic	EdgeConv	Bilinear	✓			37.32	52.07	38.51	127.91
11	✓	Dynamic	EdgeConv	Bilinear	✓		0.3	37.92	52.37	40.27	130.55
12	✓	Dynamic	EdgeConv	MLP				38.44	56.7	46.21	141.35
13	✓	Dynamic	EdgeConv	MLP	✓			34.95	53.75	41.57	130.27
14	✓	Dynamic	EdgeConv	MLP	✓		0.3	35.54	56.5	45.91	137.95
15	✓	kNN-4	EdgeConv	Bilinear				37.30	54.25	42.28	133.83
16	✓	kNN-4	EdgeConv	MLP				37.41	**57.54**	**48.71**	**143.66**
17	✓	Complete	DeepGCN	MLP	✓			37.17	55.43	42.59	135.19
18	✓	Complete	1 Cv,DeepGCN	MLP				38.35	57.11	47.12	142.58
19	✓	Complete	4 Cv,DeepGCN	MLP				37.89	**57.71**	**48.84**	**144.44**
20		Dynamic	EdgeConv	Bilinear				34.52	50.74	35.52	120.78
21		Dynamic	EdgeConv	Bilinear	✓	30		33.23	46.26	30.71	110.20
22		Dynamic	EdgeConv	Bilinear		30		32.03	50.94	37.31	120.28
23		Dynamic	EdgeConv	Bilinear	✓	30	0.3	36.89	49.02	32.13	118.04
24		Dynamic	EdgeConv	MLP				36.35	56.31	45.55	138.21
25		Dynamic	EdgeConv	MLP		30		36.54	55.72	43.97	136.22
26		Dynamic	EdgeConv	MLP		30	0.3	34.32	55.41	45.65	135.38
27		Dynamic	EdgeConv	MLP		10	0.3	35.36	54.96	44.44	134.76
28		Dynamic	EdgeConv	MLP	✓	30	0.3	34.48	55.02	42.15	131.65
29		Dynamic	EdgeConv	MLP	✓	30		35.73	54.9	41.91	132.54
30		Dynamic	EdgeConv	MLP	✓	10	0.3	34.90	56.03	44.60	135.54

Table 5.5: Variations on our model. PT (Pre-trained) = use of pre-trained ResNet and junctionness/offset branches for faster training. RF (Raw feat) = GNN applied directly to raw ResNet features (no node feature convolutions). Wait = Number of epochs where only the junction-ness and offset branches are trained before training the edge branch (default: 0). j_{thr} = junction-ness threshold used during training (default: 0.5). Sum = J-F1 + P-F1 + APLS. Cv = Convolution. *variation presented in section 5.3

Regarding the relative performance of DeepGCNs compared to shallower EdgeConv models, the graphs extracted from the images are small: they have at most $16 \times 16 = 256$ nodes and $(256) * (256 - 1)/2 = 32640$ edges. The increase in receptive field that comes with deeper models will lose effectiveness once the entire graph is covered at a given layer. Furthermore, the experiments we report with DeepGCNs in Table 5.5 are done using the complete graph as support. Messages from each node reach all other nodes in a single iteration, which reduces the benefits one can glean from using deeper models, and makes the GNNs more susceptible to the smoothing problem [LHW18]. We therefore decided to report results using shallow GNNs using the EdgeConv operator which is well suited to learning edge features and to learning on dynamic graphs. Further work will investigate using DeepGCNs on sparse graphs as well as on graphs built on larger images.

Junctionness threshold

In addition to the ablation study, we evaluated the impact of the junction-ness threshold j_{thr} used during training. Lowering this threshold seems to have a positive impact on the three metrics. Our intuition is that when the junction-ness threshold is lower, the subsequent GNN is presented with more nodes and a larger number of possible edges for each image, and is thus trained better and faster.

5.5 Limitations

Our method comes with some limitations. The first one, as said in previous sections, is the difficulty to work in very dense areas with lots of intersections. Since multiple intersections might fall into the same detection cell, our network will only be able to detect one of them, which can lead to shifted junctions. The second limitation is the inability to be trained on mask-based datasets such as DeepGlobe [Dem+18]. Our network can only be trained on graphs, and such data might not always be available. Finally, some long sections of road such as highways or bridges might lack points of interest since they do not have any junctions. Our network can miss these sections of road in the final graph as shown in Figure 5.10 and in some parts of Figure 5.1.

5.6 Other uses of our method

Our method is generic in the sense that it could be used for applications other than road extraction. In fact, it could be suitable for any image to 2D graph application. For example, our method could be used for blood vessel extraction as done in [Ven+18]. Another example is the polygonal extraction of buildings

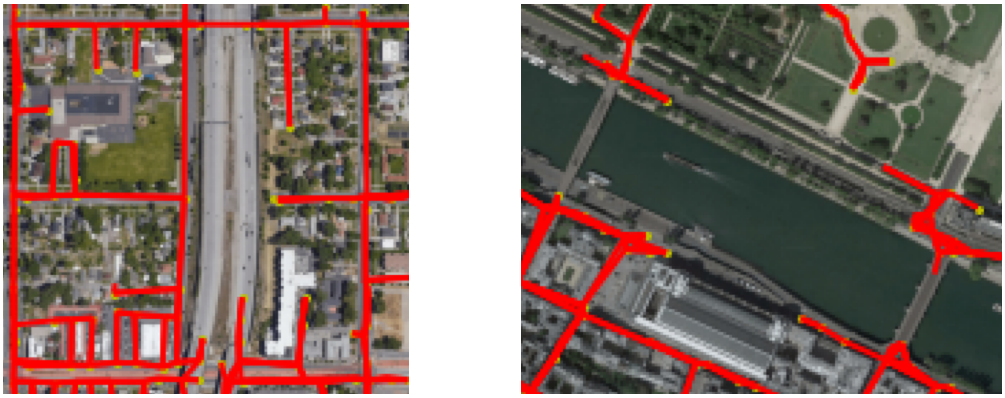


Figure 5.10: Examples of long sections of road (e.g. highways, bridges) without points of interest, which are occasionally missed by our method.

in aerial images. For this task, just like for road extraction, existing methods either rely on an iterative process [LWL19] or on post-processing of a pixel-based segmentation [LLM20]. Thus, for this task, our method is able to provide the same advantages as for road extraction. Figure 5.11 shows an example of building extraction using our method.

Figure 5.11: Our method can be used for other tasks, such as building extraction. This example is taken from the CrowdAI Mapping Challenge dataset [Moh+20]

5.7 Conclusion

We proposed a novel architecture for single-shot extraction of road graphs from satellite images. Our method first predicts sparse interest points in images using a CNN feature extractor as well as a junction detection head, and an offset regression head, trained jointly to identify candidate road intersections in a lower-resolution image and predict their position in the high-resolution image. We then combine the extracted image features and regressed point coordinates with a graph neural network to combine local and global information and infer the unknown graph structure. Our method combines aspects of graph learning and link prediction to score candidate connections between road junctions and infer the road graph. We demonstrate competitive performance with state of the art methods on three reference metrics while achieving significantly lower graph complexity and inference times (up to 91x faster than iterative models) compared to iterative and segmentation-based methods.

Even though the experimental validation of our approach was focused on road networks, our single-shot graph extraction framework can be applied to other types of problems. An example would be the extraction of blood vessels in medical images [Fra+12]. An interesting follow-up to road extraction could be to not only decide if edges exist or not in the final graph, but also infer edge attributes like road types, amount of traffic, or speed limits, as done in [Gha+21]. We also believe that our method could be adapted to tackle map update problems that have been presented in newly released datasets [BM21]. In addition, our method can further be combined with existing model compression techniques, both from the Euclidean and Geometric deep learning literature, to pave the way for on board single-shot extraction of road graphs on edge devices.

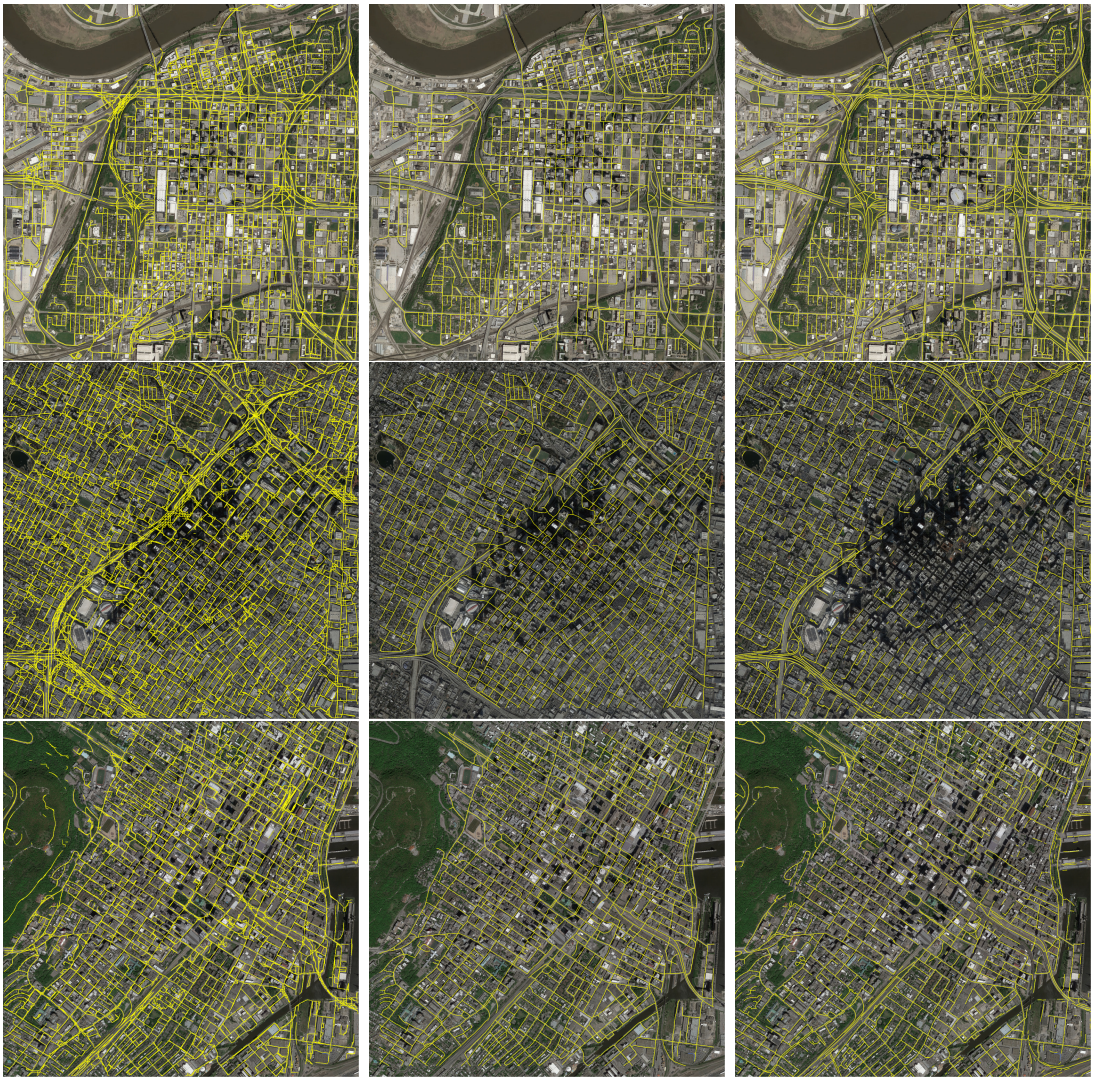


Figure 5.12: More qualitative results of our method (left) compared to RoadTracer [Bas+18] (middle) and DeepRoadMapper [MLU17] (right), on the RoadTracer test set.

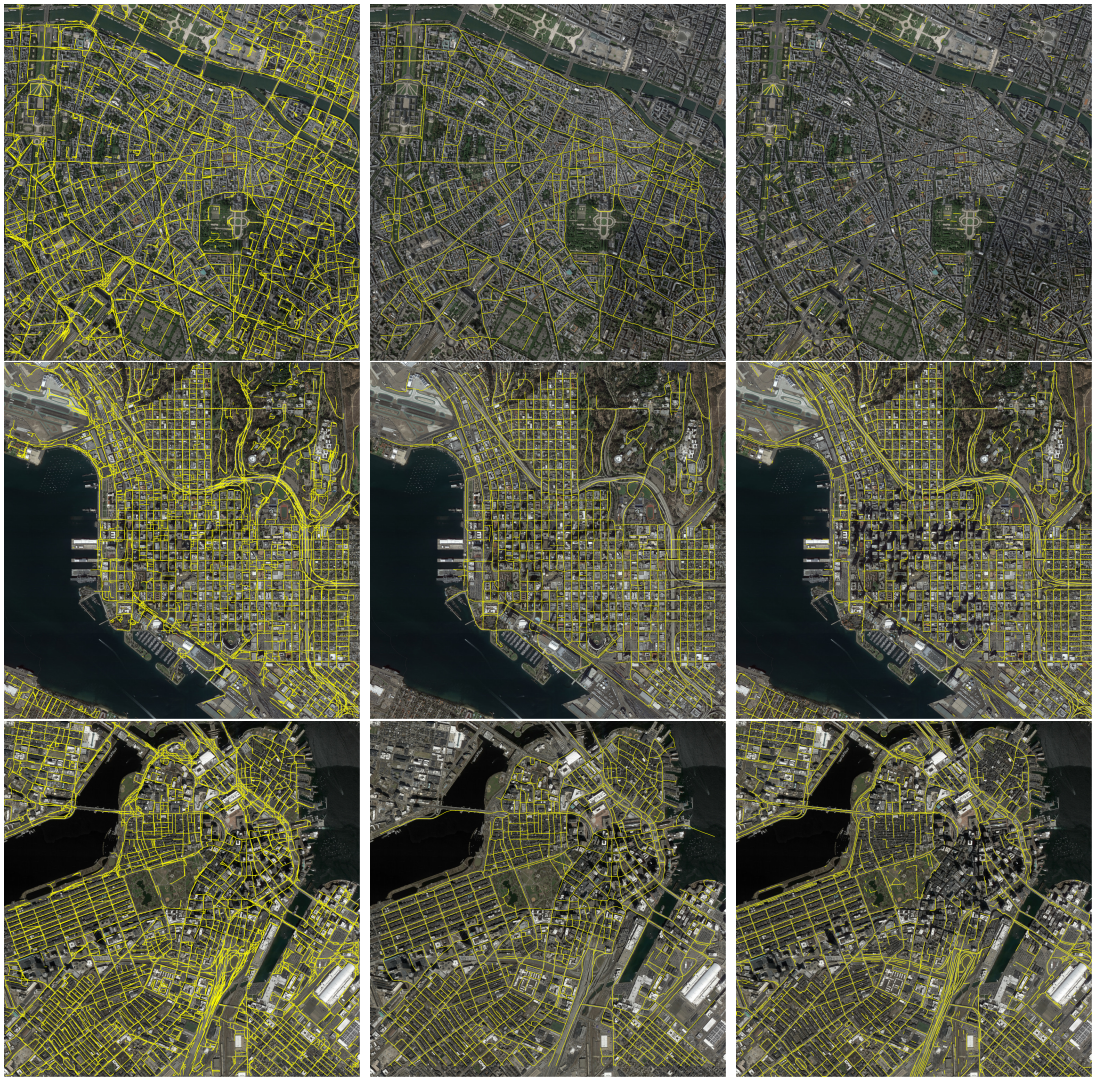


Figure 5.13: More qualitative results of our method (left) compared to RoadTracer [Bas+18] (middle) and DeepRoadMapper [MLU17] (right), on the RoadTracer test set.

Chapter 6

Conclusion and Perspectives

In this final chapter, we summarize our contributions and discuss some perspectives and ideas for future work.

6.1 Summary

The goal of this thesis was to develop Deep Learning architectures that enable onboard processing of satellite images. We chose to work on two facets of this problem. The first facet is the generally high computational requirements of state-of-the-art deep neural networks compared to the relatively low capabilities of hardware that can be included on satellites. We tackled this challenge by creating efficient architectures based on Fully Convolutional Networks. The second facet is the memory or storage limitation of edge devices, which we addressed by obtaining compact, sparse or compressible outputs. We presented two types of architectures for image segmentation, as well as a contour regression architecture, and a road graph extraction architecture.

In chapter 2, we designed fully convolutional models with a low number of parameters to address the use case of cloud segmentation on the small FPGA of OPS-SAT. As any "off-the-shelf" model from the litterature would be orders of magnitude too big for this FPGA, the main challenge was to carefully craft our architectures such that they would have a very low footprint in terms of computation, memory and number of parameters, while also being accurate. Moreover, we had to use simple operators such as classical 2D convolutions and depth-wise separable convolutions that could be easily implemented in VHDL code. We successfully trained a tiny architecture on cloud image datasets and fine-tuned it on real images from OPS-SAT as soon as they were available. The model was then implemented in VHDL and inference was performed in orbit by remotely updating the weights of the network, which was a first in this field.

Chapter 3 was dedicated to the creation of a neural network architecture that would be specifically optimized for the on-the-fly processing of images of pushbroom sensors, which is an idea that was seldom explored in this field. Being able to perform semantic analysis of images directly during the acquisition will allow a better reactivity and potentially huge storage savings, as the images would never need to be stored onboard or transmitted to the ground. It would also allow the acquitisions to run for a longer time, since the onboard storage would never be filled. The architecture that we developed has very low memory footprint and latency, and could also be used in other types of applications, such as on-the-fly character recognition in document scanners.

Next, in chapter 4, we chose to focus on the generation of object contours using a compact representation based on Fourier coefficients. The goal was to enable the creation an onboard database of objects in order perform in-flight change detection, and facilitate transmission of this database within a constellation of satellites. Notably, progressive transmission can be achieved by transmitting centroids and low frequency coefficients first, and refining the contours with high frequency coefficients later. Using geometric priors on the perimeter and simplicity of shapes, we were able to accurately represent smooth object contours with as low as 8 coefficients. The possibility of onboard processing using our method was evaluated by running our architecture on a wide array of low-power devices using an efficient backbone.

Finally, chapter 5 focused on the design of a combination of Fully Convolutional Network and Graph Neural Network for graph extraction from images, and applied it to road extraction from overhead images. We also showed that this architecture can be applied to other tasks, such as polygonal building extraction. Our novel combination of FCN and GNN allows significant computation time savings, to the point where road extraction could be done onboard. Our method generates graphs with a low complexity (*i.e.* few nodes and edges), which enables the use of subsequent graph based algorithms and GNNs onboard.

6.2 Limitations and perspectives

In this section, we provide some perspectives for future work.

General comments

The architectures developed in this thesis were designed with satellite images in mind. However, they could be used in other scenarios demanding efficient architectures and compact outputs. In the future, as processors become more efficient and more low-power AI accelerators are developed, the need for compact architectures

with compact outputs will increase. However, it is also likely that new devices will have more compute capabilities, and thus will be able to run more complex models. Nevertheless, since state-of-the-art architectures continue to grow in terms of number of parameters, there would still be a need for "smaller" networks, and techniques like quantization or pruning.

Satellite development cycles are usually long. As we start to learn about the specifications of hardware (*i.e.* sensors, storage, processors, accelerators) that will be included in the next generations of satellites, the development of neural networks tailored to these satellites will start as well. This is actually similar to what happened when we learned about the kind of hardware that would be included in OPS-SAT, and thus started developing the architectures presented in chapter 2 before its launch.

The work presented in this thesis spans a wide array of possible applications and use cases. However, this means that we could not go in as much depth as we could have for some of these applications. Notably, we missed the opportunity to develop an end-to-end pipeline for onboard change detection. While our work could certainly take part in such a pipeline, some elements are still missing. In particular, objects detected during a first image acquisition have to be precisely located in a subsequent acquisition in order to perform change detection accurately, and this is an active research topic [GCT19].

Hand-crafting of neural network architectures

With the recent advances in Neural Architecture Search (NAS) and the development of specialized toolchains and frameworks by hardware makers, we believe that the hand-crafting of neural network architectures (chapter 2) will soon be a thing of the past. Just like hand-crafting of features has been rendered obsolete by deep networks, this tedious process will not be directly done by human experts in the future. In the case of edge computing, we believe that hardware-constrained NAS will be able to find the most accurate architecture that fits within set compute, parameter and memory budgets. Similarly, FPGA makers such as Xilinx (now part of AMD) and Altera (now part of Intel) will develop toolchains that allow to port any neural network to their chips, with optimal quantization and pruning. This is also true for low-power GPUs and other accelerators. Thus, NAS and other techniques will probably grow as research areas in the coming years.

The work we did to manually design architectures for OPS-SAT was probably necessary in order to have the experiment ready for launch, and surely these architectures can be re-used for other things. Nevertheless, it is time that we could have spent developing generic NAS-based solutions that could serve other projects as well as this one.

Architectures for pushbroom sensors

The recurrent convolutional network presented in chapter 3 is not novel in the sense that ConvLSTM models have existed for quite some time. However, this is, to our knowledge, the first time that they have been used in a line-by-line segmentation task. It would be interesting to know if other tasks such as classification or object detection in images from pushbroom sensors could also be performed on-the-fly with similar or different architectures. In particular, how could the fully convolutional architectures used for detection or instance segmentation (*e.g.* chapter 4) be adapted into reccurent architectures for pushbroom sensors? It may also be that some completely different architectures could be used for these tasks.

Onboard change detection with object contours

The final goal of the contour regression architecture presented in chapter 4 was the creation of an onboard change detection database. Thus, the next development step would be to investigate how to actually perform change detection with this database. In particular, it would be interesting to perform shape signature matching in Fourier domain in order to find the objects from the database on the next image acquisition.

Polygonal building extraction with Graph Neural Networks

We showed in chapter 5 that our FCN + GNN architecture could also be used for polygonal building extraction. It would be interesting to investigate this task, and possibly modify the architecture for this. For example, the CNN used for feature extraction could be replaced by a U-Net-like architecture with upscaling in order to obtain smaller detection cells. This would probably allow a precise localization of building corners.

Regression of Graph properties

The method we showed in chapter 5 could be extended to regress node or edge attributes simultaneously as the extraction of the graph, with a few modifications to the Graph Neural Network. In the case of roads, it could be possible to regress edge properties such as road type, speed of traffic or amount of traffic. Currently, traffic alerts such as congestion or accidents are collected with smartphone applications such as Waze or Google Maps. Real-time onboard edge attribute regression could allow alerts without human intervention. Better traffic predictions could also help avoid traffic jams and their consequences.

Improving graph extraction

In a lot of fields, it is generally acceptable to give up some accuracy to gain a lot of processing speed, but while the method presented in chapter 5 is very fast, potential improvements could be investigated. We could view the offset property regressed by the offset branch of the point-of-interest detection head as a node property to be regressed by the Graph Neural Network. Since the GNN has to predict which pairs of nodes are linked, it is possible that it could be able to make a more informed decision on the spatial position of the nodes than the FCN.

Our single-head design without a Feature Pyramid made the Fully Convolutional part of the architecture really fast. We assumed that using an FPN [Lin+17a] was not necessary in our case, as the scale of satellite images does not change. We were right to some extent, since the method seems to work quite well without one. However, we did not actually try it with an FPN, and it is possible that using one could solve the long-range link prediction limitations we faced.

Map update

We believe that more combinations of CNN and GNN are going to be developed in the coming years, and that they are going to be used to solve new problems. An interesting one to solve would be the map update problem, *i.e.* change detection in maps. By externally modifying the outputs of the junction-ness and offset branches in the method presented in chapter 5, it is possible to force the existence and position of certain nodes. Thus a satellite containing a map could predict the addition or removal of edges using the GNN part of our method. An example of application would be the evacuation of populations in the case of a natural disaster (*e.g.* earthquake, flood). Roads that have been flooded or destroyed would disappear from the map in real-time and GPS apps could thus make informed decisions by knowing which routes remain available.

6.3 Publications

Papers authored during this thesis:

- **Low-power Neural Networks for Semantic Segmentation of Satellite Images.**

  Gaétan Bahl, Lionel Daniel, Matthieu Moretti, Florent Lafarge.

  ICCV Workshop on Low Power Computer Vision, 2019.

- **In space image processing using AI embedded on system on module: example of OPS-SAT cloud segmentation.**

Frédéric Férésin, Erwann Kervennic, Yves Bobichon, Edgar Lemaire, Nassim Abderrahmane, Gaétan Bahl, Ingrid Grenet, Matthieu Moretti, Michael Benguigui.

2nd European Workshop on On-Board Data Processing, 2021.

- **Binary Graph Neural Networks.**

  Mehdi Bahri, Gaétan Bahl, Stephanos Zafeiriou.

  IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), 2021.

- **Road Extraction from Overhead Images with Graph Neural Networks.**

  Gaétan Bahl, Mehdi Bahri, Florent Lafarge.

  EARTHVISION 2022 CVPR Workshop.

- **SCR: Smooth Contour Regression with Geometric Priors.**

  Gaétan Bahl, Lionel Daniel, Florent Lafarge.

  arXiv preprint arXiv:2202.03784

- **Scanner Neural Network for On-board Segmentation of Satellite Images.**

  Gaétan Bahl, Florent Lafarge.

  IGARSS 2022.

6.4 Carbon Impact Statement

While the focus of our work is to create efficient neural networks that are able to run on very low power devices, we cannot help but notice that these networks are still created and trained using power-hungry multi-GPU machines and wonder about the environmental impact. During this thesis, we missed the opportunity to track the real power consumption of our experiments, and to estimate its global environmental footprint, but we are trying as much as we can to publish these estimations, as recommended in [Hen+20; AKS20; LGI20].

In order to run the experiments required for this thesis, we estimate that we have used around 15000 GPU hours on Nvidia Tesla V100, GTX 1080Ti, RTX 2080, RTX 3090 and similar GPUs, which are rated for a power consumption of 200 to 400W. This, not counting CPUs, cooling, PSU efficiency, storage of datasets and results, amounts to 4500kWh. Since the carbon intensity of our electricity grid

is 10 gCO2/kWh, we estimate an emission of 45000 gCO2, which is equivalent to 373.8 km traveled by car according to [AKS20].

We are lucky that the carbon intensity of the electricity grid is particularly low in our country (France), so the carbon impact of the computations of this thesis is probably low compared to other works. Moreover, we can notice that in our case, there was orders of magnitude more CO2 being produced by driving to work than from our computations, even though we have spent a lot of time working from home these past two years.

List of Figures

List of Tables

Bibliography

[AB20] Hamed Alemohammad and Kevin Booth. "LandCoverNet: A global benchmark land cover classification training dataset". In: *arXiv preprint arXiv:2012.03111* (2020).

[Abd+18] Kamel Abdelouahab et al. "Accelerating CNN inference on FPGAs: A Survey". In: *ArXiv* abs/1806.01683 (2018).

[AKS20] Lasse F Wolff Anthony, Benjamin Kanding, and Raghavendra Selvan. "Carbontracker: Tracking and predicting the carbon footprint of training deep learning models". In: *arXiv:2007.03051* (2020).

[Bah+19] Gaétan Bahl et al. "Low-power neural networks for semantic segmentation of satellite images". In: *ICCV Workshops*. 2019.

[Bas+18] Favyen Bastani et al. "Roadtracer: Automatic extraction of road networks from aerial images". In: *CVPR*. 2018.

[Bat+19] Anil Batra et al. "Improved road connectivity by joint learning of orientation and segmentation". In: *CVPR*. 2019.

[BBL21] Gaetan Bahl, Mehdi Bahri, and Florent Lafarge. "Road Extraction from Overhead Images with Graph Neural Networks". In: *arXiv preprint arXiv:2112.05215* (2021).

[BBZ21] Mehdi Bahri, Gaétan Bahl, and Stefanos Zafeiriou. "Binary Graph Neural Networks". In: *CVPR*. 2021.

[BKC17] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation". In: *PAMI* 39 (2017).

[BKH16] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[BM21] Favyen Bastani and Samuel Madden. "Beyond Road Extraction: A Dataset for Map Update using Aerial Images". In: *ICCV*. 2021.

[BPS99] Lorenzo Bruzzone, Diego F Prieto, and Sebastiano B Serpico. "A neural-statistical approach to multitemporal and multisource remote-sensing image classification". In: *IEEE Transactions on Geoscience and remote Sensing* 37.3 (1999), pp. 1350–1359.

[BT19] Adrian Bulat and Georgios Tzimiropoulos. "Xnor-net++: Improved binary neural networks". In: *arXiv preprint arXiv:1909.13863* (2019).

[Cao+19] Yue Cao et al. "GCNet: Non-local Networks Meet Squeeze-Excitation Networks and Beyond". In: *ICCV Workshops*. 2019.

[Che+18] Liang-Chieh Chen et al. "Encoder-decoder with atrous separable convolution for semantic image segmentation". In: *ECCV*. 2018.

[Che+19a] Kai Chen et al. "Hybrid Task Cascade for Instance Segmentation". In: *CVPR*. 2019.

[Che+19b] Kai Chen et al. "MMDetection: Open MMLab Detection Toolbox and Benchmark". In: *arXiv:1906.07155* (2019).

[Cho17] François Chollet. "Xception: Deep learning with depthwise separable convolutions". In: *CVPR*. 2017.

[Cor+16] Marius Cordts et al. "The cityscapes dataset for semantic urban scene understanding". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 3213–3223.

[Del+07] Fabio Del Frate et al. "Use of neural networks for automatic classification from high-resolution images". In: *IEEE transactions on geoscience and remote sensing* 45.4 (2007), pp. 800–809.

[Dem+18] Ilke Demir et al. "Deepglobe 2018: A challenge to parse the earth through satellite images". In: *CVPR Workshops*. 2018.

[DKS20] Yaman Dua, Vinod Kumar, and Ravi Shankar Singh. "Comprehensive review of hyperspectral image compression algorithms". In: *Optical Engineering* 59.9 (2020), p. 090902.

[Dow05] Frederick A Downs, Roger M and Day. *National Geographic almanac of geography*. National Geographic Society, 2005.

[Drö+18] Johannes Drönner et al. "Fast cloud segmentation using convolutional neural networks". In: *Remote Sensing* 10.11 (2018).

[Eve+10] Mark Everingham et al. "The Pascal Visual Object Classes (VOC) Challenge". In: *IJCV* 88.2 (2010). ISSN: 0920-5691. DOI: 10.1007/s11263-009-0275-4. URL: http://link.springer.com/10.1007/s11263-009-0275-4.

[Fér+21] Frédéric Férésin et al. "In space image processing using AI embedded on system on module: example of OPS-SAT cloud segmentation". In: *2nd European Workshop on On-Board Data Processing*. 2021.

[Fra+12] Muhammad Moazam Fraz et al. "Blood vessel segmentation methodologies in retinal images–a survey". In: *Computer methods and programs in biomedicine* 108.1 (2012).

[Gar+18] Alberto Garcia-Garcia et al. "A survey on deep learning techniques for image and video semantic segmentation". In: *Applied Soft Computing* 70 (2018), pp. 41–65.

[GCT19] Nicolas Girard, Guillaume Charpiat, and Yuliya Tarabalka. "Noisy supervision for correcting misaligned cadaster maps without perfect ground truth data". In: *IGARSS 2019-2019 IEEE International Geoscience and Remote Sensing Symposium*. IEEE. 2019, pp. 10103–10106.

[Gha+19] Sina Ghassemi et al. "Convolutional Neural Networks for On-Board Cloud Screening". In: *Remote Sensing* 11.12 (2019). ISSN: 2072-4292. DOI: 10.3390/rs11121417. URL: https://www.mdpi.com/2072-4292/11/12/1417.

[Gha+21] Zahra Gharaee et al. "Graph representation learning for road type classification". In: *Pattern Recognition* 120 (2021).

[Gil+17] Justin Gilmer et al. "Neural message passing for quantum chemistry". In: *ICML*. 2017.

[Gir+14] Ross Girshick et al. "Rich feature hierarchies for accurate object detection and semantic segmentation". In: *CVPR*. 2014.

[Gir15] Ross Girshick. "Fast r-cnn". In: *ICCV*. 2015.

[GL21] Vivien Sainte Fare Garnot and Loic Landrieu. "Panoptic Segmentation of Satellite Image Time Series with Convolutional Temporal Attention Networks". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 4872–4881.

[GLL19] Golnaz Ghiasi, Tsung-Yi Lin, and Quoc V Le. "Nas-fpn: Learning scalable feature pyramid architecture for object detection". In: *CVPR*. 2019.

[Gon+20] Shunwang Gong et al. "Geometrically principled connections in graph neural networks". In: *CVPR*. 2020.

[Guo+18] Yanming Guo et al. "A review of semantic segmentation using deep neural networks". In: *International journal of multimedia information retrieval* 7.2 (2018).

[Ham18] Muhammad K A Hamdan. "VHDL auto-generation tool for optimized hardware acceleration of convolutional neural networks on FPGA (VGT)". PhD thesis. Iowa State University, 2018. DOI: 10.31274/etd-180810-5998. URL: https://lib.dr.iastate.edu/etd/16368/.

[He+16] Kaiming He et al. "Deep residual learning for image recognition". In: *CVPR*. 2016.

[He+17] Kaiming He et al. "Mask r-cnn". In: *ICCV*. 2017.

[Hen+20] Peter Henderson et al. "Towards the systematic reporting of the energy and carbon footprints of machine learning". In: *Journal of Machine Learning Research* 21.248 (2020).

[How+17] Andrew G Howard et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications". In: *arXiv:1704.04861* (2017).

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. "Long short-term memory". In: *Neural computation* 9.8 (1997), pp. 1735–1780.

[HSD73] Robert M Haralick, Karthikeyan Shanmugam, and Its' Hak Dinstein. "Textural features for image classification". In: *IEEE Transactions on systems, man, and cybernetics* 6 (1973), pp. 610–621.

[Hua11] Bormin Huang. *Satellite data compression*. Springer Science & Business Media, 2011.

[Hub+17] Itay Hubara et al. "Quantized neural networks: Training neural networks with low precision weights and activations". In: *The Journal of Machine Learning Research* 18.1 (2017), pp. 6869–6898.

[Hug+14] M. Hughes et al. "Automated Detection of Cloud and Cloud Shadow in Single-Date Landsat Imagery Using Neural Networks and Spatial Post-Processing". In: *Remote Sensing* 6.6 (2014). ISSN: 2072-4292. DOI: 10.3390/rs6064907. URL: http://www.mdpi.com/2072-4292/6/6/4907.

[HVD15] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network". In: *arXiv preprint arXiv:1503.02531* (2015).

[Jac+18] Benoit Jacob et al. "Quantization and training of neural networks for efficient integer-arithmetic-only inference". In: *CVPR*. 2018.

[Jet+17] Saumya Jetley et al. "Straight to Shapes: Real-Time Detection of Encoded Shapes". In: *CVPR*. 2017.

[Jia+09] Jia Deng et al. "ImageNet: A large-scale hierarchical image database". In: *CVPR*. 2009. ISBN: 978-1-4244-3992-8. DOI: 10.1109/CVPRW.2009.5206848.

[JTP21] Uthayakumar Jayasankar, Vengattaraman Thirumal, and Dhavachelvan Ponnurangam. "A survey on data compression techniques: From the perspective of data quality, coding schemes, data type and applications". In: *Journal of King Saud University-Computer and Information Sciences* 33.2 (2021), pp. 119–140.

[Kaz+20] Anees Kazi et al. "Differentiable graph module (DGM) for graph convolutional networks". In: *arXiv preprint arXiv:2002.04999* (2020).

[KB14] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[KFS15] Nikolaos Karianakis, Thomas J. Fuchs, and Stefano Soatto. "Boosting Convolutional Features for Robust Object Proposals". In: (2015). arXiv: 1503.06350. URL: https://www.cv-foundation.org/openaccess/content%7B%5C_%7Dcvpr%7B%5C_%7D2015/html/Long%7B%5C_%7DFully%7B%5C_%7DConvolutional%7B%5C_%7DNetworks%7B%5C_%7D2015%7B%5C_%7DCVPR%7B%5C_%7Dpaper.html%20http://arxiv.org/abs/1503.06350.

[Kip+18] Thomas Kipf et al. "Neural relational inference for interacting systems". In: *ICML*. 2018.

[KP18] Andreas Kamilaris and Francesc X Prenafeta-Boldú. "Deep learning in agriculture: A survey". In: *Computers and electronics in agriculture* 147 (2018).

[KSH12] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *NeurIPS* (2012).

[KVW19] Wouter Kool, Herke Van Hoof, and Max Welling. "Stochastic beams and where to find them: The gumbel-top-k trick for sampling sequences without replacement". In: *ICML*. 2019.

[KW16] Thomas N Kipf and Max Welling. "Variational graph auto-encoders". In: *arXiv preprint arXiv:1611.07308* (2016).

[KW17] Thomas N Kipf and Max Welling. "Semi-Supervised Classification with Graph Convolutional Neural Networks". In: *ICLR*. 2017. ISBN: 2004012439. DOI: 10.1051/0004-6361/201527329. arXiv: 1803.01118. URL: http://arxiv.org/abs/1803.01118.

[Lam+18] Darius Lam et al. "xview: Objects in context in overhead imagery". In: *arXiv preprint arXiv:1802.07856* (2018).

[LDS90] Yann Lecun, J S Denker, and Sara A. Solla. "Optimal Brain Damage". In: *NIPS*. 1990.

[LeC+98] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998).

[LGI20] Loic Lannelongue, Jason Grealey, and Michael Inouye. "Green Algorithms: Quantifying the carbon emissions of computation". In: *arXiv:2007.07610* (2020).

[LH20] Renbao Lian and Liqin Huang. "DeepWindow: Sliding window based on deep learning for road extraction from remote sensing images". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020). ISSN: 21511535. DOI: 10.1109/JSTARS.2020.2983788.

[LHW18] Qimai Li, Zhichao Han, and Xiao Ming Wu. "Deeper insights into graph convolutional networks for semi-supervised learning". In: *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*. 2018. ISBN: 9781577358008. arXiv: 1801.07606.

[Li+16] Hao Li et al. "Pruning filters for efficient convnets". In: *ArXiv:1608.08710* (2016).

[Li+19a] Guohao Li et al. "DeepGCNs: Can GCNs go as deep as CNNs?" In: *ICCV*. 2019. ISBN: 9781728148038. DOI: 10.1109/ICCV.2019.00936. arXiv: 1904.03751.

[Li+19b] Guohao Li et al. "Deepgcns: Can gcns go as deep as cnns?" In: *ICCV*. 2019.

[Li+20] Guohao Li et al. "Deepergcn: All you need to train deeper gcns". In: *arXiv preprint arXiv:2006.07739* (2020).

[Lia+20] Renbao Lian et al. "Road extraction methods in high-resolution remote sensing images: A comprehensive review". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020). DOI: 10.1109/JSTARS.2020.3023549.

[Lin+14] Tsung-Yi Lin et al. "Microsoft COCO: Common objects in context". In: *ECCV*. 2014.

[Lin+17a] Tsung-Yi Lin et al. "Feature pyramid networks for object detection". In: *CVPR*. 2017.

[Lin+17b] Tsung-Yi Lin et al. "Focal loss for dense object detection". In: *ICCV*. 2017.

[Liu+16] Wei Liu et al. "Ssd: Single shot multibox detector". In: *ECCV*. 2016.

[Liu+18] Shu Liu et al. "Path Aggregation Network for Instance Segmentation". In: *CVPR*. 2018.

[Liu+19] Cheng-Chien Liu et al. "Clouds Classification from Sentinel-2 Imagery with Deep Residual Learning and Semantic Image Segmentation". In: *Remote Sensing* 11.2 (2019).

[Liu+21] Zichen Liu et al. "DANCE: A Deep Attentive Contour Model for Efficient Instance Segmentation". In: *WACV*. 2021.

[LLM20] Muxingzi Li, Florent Lafarge, and Renaud Marlet. "Approximating shapes in images with low-complexity polygons". In: *CVPR*. 2020.

[Lon+21] Yang Long et al. "On creating benchmark dataset for aerial image interpretation: Reviews, guidances, and million-aid". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 14 (2021), pp. 4205–4230.

[LSD15] Jonathan Long, Evan Shelhamer, and Trevor Darrell. "Fully convolutional networks for semantic segmentation". In: *CVPR*. 2015.

[LTA16] Darryl Lin, Sachin Talathi, and Sreekanth Annapureddy. "Fixed point quantization of deep convolutional networks". In: *ICML*. 2016.

[LWL19] Zuoyue Li, Jan Dirk Wegner, and Aurelien Lucchi. *Topological Map Extraction From Overhead Images*. 2019. URL: http://openaccess.thecvf.com/content%7B%5C_%7DICCV%7B%5C_%7D2019/html/Li%7B%5C_%7DTopological%7B%5C_%7DMap%7B%5C_%7DExtraction%7B%5C_%7DFrom%7B%5C_%7DOverhead%7B%5C_%7DImages%7B%5C_%7DICCV%7B%5C_%7D2019%7B%5C_%7Dpaper.html.

[Mag+17] Emmanuel Maggiori et al. "Can Semantic Labeling Methods Generalize to Any City? The Inria Aerial Image Labeling Benchmark". In: *IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE. 2017.

[MBZ20] Hamd ul Moqeet Riaz, Nuri Benbarka, and Andreas Zell. "FourierNet: Compact Mask Representation for Instance Segmentation Using Differentiable Shape Decoders". In: *ICPR*. 2020.

[Meh+18] Sachin Mehta et al. "Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation". In: *ECCV*. 2018.

[MH10] Volodymyr Mnih and Geoffrey E. Hinton. "Learning to Detect Roads in High-Resolution Aerial Images". In: *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)* 6316 LNCS (PART 6 2010). DOI: 10.1007/978-3-642-15567-3_16. URL: https://link.springer.com/chapter/10.1007/978-3-642-15567-3_16.

[MHT18] Giorgio Morales, Samuel G Huamán, and Joel Telles. "Cloud Detection in High-Resolution Multispectral Satellite Imagery Using Deep Learning". In: *ICANN*. 2018.

[MIO11] Giorgos Mountrakis, Jungho Im, and Caesar Ogole. "Support vector machines in remote sensing: A review". In: *ISPRS Journal of Photogrammetry and Remote Sensing* 66.3 (2011), pp. 247–259.

[MKS18] S. Mohajerani, T. A. Krammer, and P. Saeedi. ""A Cloud Detection Algorithm for Remote Sensing Images Using Fully Convolutional Neural Networks"". In: *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*. Aug. 2018, pp. 1–5. DOI: 10.1109/MMSP.2018.8547095.

[MLU17] Gellert Mattyus, Wenjie Luo, and Raquel Urtasun. "DeepRoadMapper: Extracting Road Topology from Aerial Images". In: *ICCV* (2017). DOI: 10.1109/ICCV.2017.372.

[Mni13] Volodymyr Mnih. *Machine learning for aerial image labeling*. University of Toronto (Canada), 2013.

[Moh+20] Sharada Prasanna Mohanty et al. "Deep Learning for Understanding Satellite Imagery: An Experimental Survey". In: *Frontiers in Artificial Intelligence* 3 (2020).

[Mol+16] Pavlo Molchanov et al. "Pruning convolutional neural networks for resource efficient inference". In: *ArXiv:1611.06440* (2016).

[Mot+14] Roozbeh Mottaghi et al. "The role of context for object detection and semantic segmentation in the wild". In: *CVPR*. 2014. ISBN: 9781479951178. DOI: 10.1109/CVPR.2014.119. URL: http://host.robots.ox.ac.uk:8080/.

[MS19] S. Mohajerani and P. Saeedi. "Cloud-Net: An End-To-End Cloud Detection Algorithm for Landsat 8 Imagery". In: *IGARSS 2019 - 2019 IEEE International Geoscience and Remote Sensing Symposium*. July 2019, pp. 1029–1032. DOI: 10.1109/IGARSS.2019.8898776.

[NHH15] Hyeonwoo Noh, Seunghoon Hong, and Bohyung Han. "Learning deconvolution network for semantic segmentation". In: *ICCV*. 2015. ISBN: 9781467383912. DOI: 10.1109/ICCV.2015.178. arXiv: 1505.04366. URL: http://arxiv.org/abs/1505.04366.

[Ohl18] Sébastien Ohleyer. "Building segmentation on satellite images". In: *Web: https://project. inria. fr/aerialimagelabeling/files/2018/01/fp_ ohleyer_ compressed. pdf* (2018).

[Pas+19] Adam Paszke et al. "PyTorch: An Imperative Style, High-Performance Deep Learning Library". In: *NeurIPS*. Ed. by H. Wallach et al. 2019. URL: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[Pen+20] Sida Peng et al. "Deep snake for real-time instance segmentation". In: *CVPR*. 2020.

[Rav+20] Nikhila Ravi et al. "Accelerating 3D Deep Learning with PyTorch3D". In: *arXiv:2007.08501* (2020).

[Red+16] Joseph Redmon et al. "You only look once: Unified, real-time object detection". In: *CVPR*. 2016.

[Ren+15] Shaoqing Ren et al. "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks". In: *NeurIPS*. 2015.

[RF18] Joseph Redmon and Ali Farhadi. "Yolov3: An incremental improvement". In: *arXiv:1804.02767* (2018).

[RFB15] 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*. 2015.

[RW73] P Ready and P Wintz. "Information extraction, SNR improvement, and data compression in multispectral imagery". In: *IEEE Transactions on communications* 21.10 (1973), pp. 1123–1131.

[SB91] Michael J Swain and Dana H Ballard. "Color indexing". In: *International journal of computer vision* 7.1 (1991), pp. 11–32.

[SC19] Renee Su and Rong Chen. "Land cover change detection via semantic segmentation". In: *arXiv preprint arXiv:1911.12903* (2019).

[Sca+09] Franco Scarselli et al. "The graph neural network model." In: *IEEE transactions on neural networks* 20.1 (2009). ISSN: 1941-0093. DOI: 10.1109/TNN.2008.2005605.

[Sch+18] Michael Schlichtkrull et al. "Modeling relational data with graph convolutional networks". In: *European semantic web conference*. Springer. 2018.

[She+20] Mohammadreza Sheykhmousa et al. "Support vector machine versus random forest for remote sensing image classification: A meta-analysis and systematic review". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 13 (2020), pp. 6308–6325.

[She+21] Xing Shen et al. "DCT-Mask: Discrete Cosine Transform Mask Representation for Instance Segmentation". In: *CVPR*. 2021.

[Sia+18] Mennatullah Siam et al. "RTSeg: Real-Time Semantic Segmentation Comparative Study". In: *ICIP* 2 (2018). ISSN: 15224880. DOI: 10.1109/ICIP.2018.8451495.

[Sif14] L. Sifre. "Rigid-motion scattering for image classification". PhD thesis. 2014.

[Sin] Sinergise. *Modified Copernicus Sentinel data 2017/Sentinel Hub*. https://sentinel-hub.com/.

[SZ14] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition". In: *ArXiv:1409.1556* (2014).

[Sze+15] Christian Szegedy et al. "Going deeper with convolutions". In: *CVPR*. 2015.

[Tan+20] Yong-Qiang Tan et al. "Vecroad: Point-based iterative graph exploration for road graphs extraction". In: *CVPR*. 2020.

[TGH08] Andrew J Tatem, Scott J Goetz, and Simon I Hay. "Fifty years of earth observation satellites: Views from above have lead to countless advances on the ground in both scientific knowledge and daily life". In: *American scientist* 96.5 (2008), p. 390.

[Tia+19] Zhi Tian et al. "Fcos: Fully convolutional one-stage object detection". In: *ICCV*. 2019.

[TL19] Mingxing Tan and Quoc Le. "Efficientnet: Rethinking model scaling for convolutional neural networks". In: *ICML*. 2019.

[Ven+18] Carles Ventura et al. "Iterative Deep Learning for Road Topology Extraction". In: *BMVC* (2018). URL: https://arxiv.org/abs/1808.09814v1.

[VLB18] Adam Van Etten, Dave Lindenbaum, and Todd M Bacastow. "Spacenet: A remote sensing dataset and challenge series". In: *arXiv preprint arXiv:1807.01232* (2018).

[Wan+19a] Jiaqi Wang et al. "Carafe: Content-aware reassembly of features". In: *ICCV*. 2019.

[Wan+19b] Yue Wang et al. "Dynamic Graph CNN for Learning on Point Clouds". In: *ACM Trans. Graph.* 38.5 (Oct. 2019). ISSN: 0730-0301. DOI: 10.1145/3326362. URL: http://doi.acm.org/10.1145/3326362.

[Wan+20a] Hanchen Wang et al. *Binarized Graph Neural Network*. Tech. rep. 2020. arXiv: 2004.11147v1. URL: http://snap.stanford.edu/proj/embeddings-www.

[Wan+20b] Xinlong Wang et al. "Solo: Segmenting objects by locations". In: *ECCV*. 2020.

[Wan+20c] Xinlong Wang et al. "Solov2: Dynamic, faster and stronger". In: *arXiv:2003.10152* (2020).

[Wu+18] Shuang Wu et al. "Training and inference with integers in deep neural networks". In: *ArXiv:1802.04680* (2018).

[Xie+17] Saining Xie et al. "Aggregated residual transformations for deep neural networks". In: *CVPR*. Vol. 2017-Janua. 2017, pp. 5987–5995. ISBN: 9781538604571. DOI: 10.1109/CVPR.2017.634. arXiv: arXiv:1611.05431v2. URL: http://openaccess.thecvf.com/content%7B%5C_%7Dcvpr%7B%5C_%7D2017/html/Xie%7B%5C_%7DAggregated%7B%5C_%7DResidual%7B%5C_%7DTransformations%7B%5C_%7DCVPR%7B%5C_%7D2017%7B%5C_%7Dpaper.html.

[Xie+20] Enze Xie et al. "Polarmask: Single shot instance segmentation with polar representation". In: *CVPR*. 2020.

[Xin+15] Shi Xingjian et al. "Convolutional LSTM network: A machine learning approach for precipitation nowcasting". In: *Advances in neural information processing systems*. 2015, pp. 802–810.

[XMS19] Yunyang Xiong, Ronak Mehta, and Vikas Singh. "Resource constrained neural network architecture search: Will a submodularity assumption help?" In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2019, pp. 1901–1910.

[Xu+19a] Wenqiang Xu et al. "Explicit shape encoding for real-time instance segmentation". In: *ICCV*. 2019.

[Xu+19b] Yonghao Xu et al. "Advanced multi-sensor optical remote sensing for urban land use and land cover classification: Outcome of the 2018 IEEE GRSS data fusion contest". In: *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 12.6 (2019), pp. 1709–1724.

[Yan+14] Bishan Yang et al. "Embedding entities and relations for learning and inference in knowledge bases". In: *arXiv preprint arXiv:1412.6575* (2014).

[YK15] Fisher Yu and Vladlen Koltun. "Multi-scale context aggregation by dilated convolutions". In: *ArXiv:1511.07122* (2015).

[YVS09] Guoxia Yu, Tanya Vladimirova, and Martin N Sweeting. "Image compression systems on board satellites". In: *Acta Astronautica* 64.9-10 (2009), pp. 988–1005.

[ZC18] Muhan Zhang and Yixin Chen. "Link Prediction Based on Graph Neural Networks". In: *NeurIPS*. Ed. by S Bengio et al. 2018. URL: https://proceedings.neurips.cc/paper/2018/file/53f0d7c537d99b3824f0f99d
Paper.pdf.

[ZGC02] Ying Zhang, B Guindon, and Josef Cihlar. "An image transform to characterize and compensate for spatial variations in thin cloud contamination of Landsat images". In: *Remote Sensing of Environment* 82 (2002).

[Zha+18] Xiangyu Zhang et al. "ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices". In: *CVPR*. 2018. ISBN: 9781538664209. DOI: 10.1109/CVPR.2018.00716.

[Zha+20] Rufeng Zhang et al. "Mask Encoding for Single Shot Instance Segmentation". In: *CVPR*. 2020.

[Zhu+17] Xiao Xiang Zhu et al. "Deep learning in remote sensing: A comprehensive review and list of resources". In: *IEEE Geoscience and Remote Sensing Magazine* 5.4 (2017), pp. 8–36.

[Zhu+19] Xizhou Zhu et al. "Deformable convnets v2: More deformable, better results". In: *CVPR*. 2019.

[ZLW18] Zhengxin Zhang, Qingjie Liu, and Yunhong Wang. "Road extraction by deep residual u-net". In: *IEEE Geoscience and Remote Sensing Letters* 15.5 (2018).

[ZWW15] Zhe Zhu, Shixiong Wang, and Curtis E Woodcock. "Improvement and expansion of the Fmask algorithm: Cloud, cloud shadow, and snow detection for Landsats 4–7, 8, and Sentinel 2 images". In: *Remote Sensing of Environment* 159 (2015).

[ZXS18] Zhaoxiang Zhang, Guodong Xu, and Jianing Song. "CubeSat cloud detection based on JPEG2000 compression and deep learning". In: *Advances in Mechanical Engineering* 10 (2018).

[ZZW18] Lichen Zhou, Chuang Zhang, and Ming Wu. "D-linknet: Linknet with pretrained encoder and dilated convolution for high resolution satellite imagery road extraction". In: *CVPR Workshops*. 2018.