DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Urban change detection on satellites using deep learning

A case of moving AI into space for improved Earth observation

OLIVER PETRI

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Urban change detection on satellites using deep learning

A case of moving AI into space for improved Earth observation

Author

Oliver Petri <opetri@kth.se, oliver.ce.petri@gmail.com>
Degree Programme in Computer Science and Engineering,
Master's Programme, Machine Learning, 120 credits
KTH Royal Institute of Technology

Examiner

Markus Flierl
KTH Royal Institute of Technology

Supervisors

Elisabeth Schold Linnér
Unibap AB (publ)

Atsuto Maki
KTH Royal Institute of Technology

Host company

Unibap AB (publ)
Uppsala, Sweden

Abstract

Change detection using satellite imagery has applications in urban development, disaster response and precision agriculture. Current deep learning models show promising results. However, on-board computers are typically highly constrained which poses a challenge for deployment. On-board processing is desirable for saving bandwidth by downlinking only novel and valuable data.

The goal of this work is to determine what change detection models are most technically feasible for on-board use in satellites. The novel patch based model MobileGoNogo is evaluated along current state-of-the-art models. Technical feasibility was determined by observing accuracy, inference time, storage buildup, memory usage and resolution on a satellite computer tasked with detecting changes in buildings from the SpaceNet 7 dataset. Three high level approaches were taken; direct classification, post classification and patch-based change detection.

None of the models compared in the study fulfilled all requirements for general technical feasibility. Direct classification models were highly resource intensive and slow. Post classification model had critically low accuracy but desirable storage characteristics. Patch based MobileGoNogo performed better by all metrics except in resolution where it is significantly lower than any other model. We conclude that the novel model offers a feasible solution for low resolution, non-critical applications.

Keywords

machine learning, deep learning, change detection, remote sensing, siamese CNN, on-board data processing, satellites

Abstract

Upptäckt av förändringar med hjälp av satellitbilder har tillämpningar inom bl.a. stadsutveckling, katastrofinsatser och precisionsjordbruk. De nuvarande modellerna för djupinlärning visar lovande resultat. Datorerna ombord satelliter är dock vanligtvis mycket begränsade, vilket innebär en utmaning för användningen av dessa modeller. Databehandling ombord är önskvärd för att spara bandbredd genom att endast skicka ner nya och värdefulla data.

Målet med detta arbete är att fastställa vilka modeller för upptäckt av förändringar som är mest tekniskt genomförbara för användning ombord på satelliter. Den nya bildfältbaserade modellen MobileGoNogo utvärderas tillsammans med de senaste modellerna. Den tekniska genomförbarheten fastställdes genom att observera träffsäkerhet, inferenstid, lagring, minnesanvändning och upplösning på en satellitdator med uppgift att upptäcka förändringar i byggnader från SpaceNet 7-dataset. Tre tillvägagångssätt på hög nivå användes: direkt klassificering, postklassificering och fältbaserad klassificering.

Ingen av de modeller som jämfördes i studien uppfyllde alla krav på allmän teknisk genomförbarhet. Direkta klassificeringsmodeller var mycket resurskrävande och långsamma. Postklassificeringsmodellen hade kritiskt låg träffsäkerhet men önskvärda lagringsegenskaper. Den bildfältbaserade MobileGoNogo-modellen var bättre i alla mätvärden utom i upplösningen, där den var betydligt lägre än någon annan modell. Vi drar slutsatsen att den nya modellen erbjuder en genomförbar lösning för icke-kritiska tillämpningar med låg upplösning.

Nyckelord

maskininlärning, djupinlärning, förändringsdetektion, fjärranalys, siamesiska CNN, databehandling ombord, satelliter

Acknowledgements

I would like to thank my supervisor Elisabeth Schold Linnér at industry partner Unibap AB for consistent dedication and support throughout the project. Furthermore I would like to thank Oskar Flordal and the rest of Unibap for believing in my idea and providing resources to enable this work.

I would also like to thank my academic supervisor Atsuto Maki and examiner Markus Flierl at KTH Royal Institute of Technology for all the work and support.

Finally, I would like to thank my partner Madeline Clarke for your listening and support.

Figure 2.3.1 is used with permission from Hongruixuan Chen of Wuhan University.

Oliver Petri

November 6, 2021

Acronyms

CPU Central Processing Unit
GPU Graphical Processing Unit
FPGA Field Programmable Gate Array
CD Change Detection
OBC On-board Computer
ML Machine Learning
SAR Synthetic Aperture Radar
RS Remote Sensing
EO Earth Observation
CNN Convolutional Neural Network
GAN Generative Adversarial Network
DBN Deep Belief Network
AE Auto Encoder
RNN Recurrent Neural Network
AI Artificial Intelligence
SWaP Size, Weight and Power
COTS Commercial-off-the-shelf
ESA European Space Agency
VPU Visual Processing Unit
SoC System on Chip
HSA Heterogeneous System Architecture
SOTA State-of-the-art
OS Operating System
TF TensorFlow
TFLite TensorFlow Lite
FCN Fully Convolutional Network

TDP Thermal Design Power

AOI Area of Interest

DL Deep Learning

SD Standard Deviation

OSCD Onera Satellite Change Detection

SN7 SpaceNet 7

BCE Binary Cross Entropy

VHR Very High Resolution

FCSD Fully Convolutional Siamese Difference

FCSC Fully Convolutional Siamese Concatenated

FCEF Fully Convolutional Early Fusion

AUC Area Under Curve

PRC Precision-Recall Curve

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter describes the specific problem that this thesis addresses, the context of the problem, the goals of this thesis project, and outlines the structure of the thesis.

We investigate the use of Machine Learning (ML) based change detection models on satellite computers. This thesis is concerned with changes in urban structures. Using ML for change detection in satellite images has been studied extensively [22], however it has not been studied in the context of running directly on satellite computers.

1.1 Background

In recent years, there has been a rapid growth in small- and nanosatellite markets [9] for various use cases such as Earth observation. With an increasing number of satellites as well as higher resolution image sensors, processing and downlinking the increasing data volumes pose some challenges. Intelligent processing of sensor data on board the satellite has the potential to reduce the downlink bottleneck by only transmitting novel data of high value. Change detection using machine learning is one such kind of processing. For Earth observation tasks such as precision agriculture, disaster monitoring, and urban development monitoring, only images with significant changes of a certain kind may be of interest. In disaster monitoring, low latency is also of importance, which on-board processing enables. With on-board computing becoming more capable and accessible recently [9] it is feasible to do on-board image processing on some computers and the capability is expected to increase. Unibap AB develops satellite computing hardware, enabling their SpaceCloud service which

allows customers to deploy ML workloads on satellites. Change detection could be an important part of some ML pipelines deployed to SpaceCloud and it is therefore of interest for Unibap to know what models would be most suitable to use for various use cases.

Change Detection (CD) is an active area of research in ML with recent developments exploring various machine learning models such as siamese Convolutional Neural Network (CNN), Recurrent Neural Network (RNN), Generative Adversarial Network (GAN), and autoencoders for supervised and unsupervised learning [22]. A satellite computing platform or constellation thereof puts certain constraints on for example resolution, storage, speed, bandwidth, and relevant use cases which influence the choice of model.

1.2 Problem

Which ML models are most suitable for urban change detection in high-performance on-board computers for satellites, for example, Unibap's iX5?

1.3 Purpose

The purpose of the thesis is to enable and improve change detection systems by applying them directly on satellites. Better change detection can benefit a range of actors assessing population growth or early warning systems for natural disasters for example.

1.4 Goal

The goal of this work is to provide comprehensive data and analysis of some CD methods run on an On-board Computer (OBC). As a result, Unibap AB and others in the industry can evaluate technical feasibility of change detection models for their OBC and identify areas of further research.

For the academic audience, the thesis bridges the gap of knowledge between ML for CD and ML for OBC as illustrated in Figure 1.4.1.

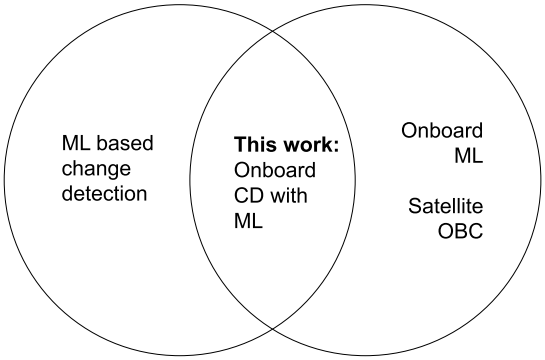


Figure 1.4.1: Domain of this work as intersection of adjacent fields

1.5 Benefits, Ethics and Sustainability

Change detection is essential to monitoring tasks, including surveillance satellites. In the studied use cases, mass surveillance of people will not be included for privacy reasons. Experiments will not be data-intensive enough to have a significant impact in terms of energy usage and compute resources. The sustainability of change detection tasks in satellite images may increase by moving to on-board processing. A major goal of on-board processing is to reduce the volume of data downlinked, leading to energy savings and a more sustainable use of available bandwidth.

1.6 Methodology

We carry out a comparative study in which a number of deep learning CD models are studied through runtime experiments and theoretical analysis. We further propose a novel model called MobileGoNogo which is compared to current State-of-the-art (SOTA) models. Quantitative analysis of collected data is performed along with an evaluation of technical feasibility.

The method follows steps:

1. CD method survey and selection
2. Model training on dataset SpaceNet 7 [8] as described in section 3.3
3. Experimentally benchmark an urban change detection task on a CPU and GPU similar to that in satellite missions, see section 4.1
4. Theoretical limitations and requirements are analyzed in sections 4.3 through 4.5

5. Novel CD method development in section 3.2.1, repeat steps 2-4
6. Technical feasibility is evaluated using observed data, modeled scenarios and available hardware in section 4.6

We compare model by runtime analysis of change detection experiments with implementations of different kinds of models. Quantitative metrics observed are accuracy (F1, precision, recall), inference time, RAM, and storage usage as they were deemed relevant to the evaluation. Experiments were run on Unibap's satellite computers on the ground. We model realistic scenarios to evaluate the system against potential use cases. TensorFlow models were used and modified to run as TensorFlow Lite models on the hardware for optimized storage and performance on embedded Linux devices (e.g. satellite computers). Finally, the practicality of the system was evaluated in relation to selected use cases and hardware platforms.

In particular, tests focused on semantic change detection of buildings since it has been identified as a useful and commonly researched application of change detection lately.

1.7 Stakeholders

Unibap AB
KTH Royal Institute of Technology

1.8 Delimitations

Delimitations of the project are:

- Only TensorFlow (Lite) was used as ML framework. This criterion limits model selection. Other ML frameworks may give different results.
- Only widely available open-source models were used for reproducibility.
- Only optical sensing data is used. SAR and hyperspectral sensors are common and used with change detection methods not included in this work.
- Urban development changes, i.e. buildings, are the only changes detected. Land-cover changes, for example, may work better with models not used here.

- Radiation effects and tolerance are not considered.
- A detailed energy consumption analysis is not performed.

1.9 Outline

Chapter 2 gives a background on relevant research for change detection, satellite computers and their limitations. In Chapter 3 the compared models, dataset processing, novel model design and practical description of experiments is described. Results from runtime experiments and theoretical analysis are presented in Chapter 4. Finally, conclusions and discussions of the results are presented in Chapter 5 to answer the question of which method is most technically feasible.

Chapter 2

Background

This chapter introduces the areas of satellite remote sensing, computers, on-board processing an ML, change detection, and related ML background as well as related work.

2.1 Satellite Remote Sensing

Satellites have been used for observing Earth since the first orbital satellite Sputnik 1. Earth Observation (EO) satellites provide reliable and repeated coverage of data collection. Analyzing the data, valuable information about the state and evolution of the planet can be extracted for various applications. A few examples are natural disasters, deforestation, and urban growth.

Common sensors are Synthetic Aperture Radar (SAR), optical, and hyperspectral imaging sensors. Developments in sensor technology drive payload size to become smaller [26].

Satellites in sun-synchronous orbits revisit any given area at the same time of day year-round, minimizing changes due to lighting between images. Consistent sensing aids analysis of the imagery, such as when using change detection methods. Other satellites may be placed in geo-synchronous orbit such that it points at the same area at all times, allowing high temporal resolution but poor coverage and low imaging resolution [7].

Revisit time, or temporal resolution, is the time period between sensing of an area. A decrease in revisit time can be achieved with constellations of satellites imaging the

same areas. With the decrease in the cost of small satellites such as CubeSats, some missions use large (100+) constellations of satellites to achieve faster revisit times [26].

2.2 Satellite data processing

Processing the collected data is an important step to make it usable and valuable. The European Space Agency (ESA) Φ-Lab emphasizes the need to analyze the large amounts of data coming from new EO satellites using Artificial Intelligence (AI) to bring ”new insight and predictive capabilities” [15]. Where and how to best process the data with AI depends on factors such as satellite capabilities, bandwidth requirements, and application.

Sensor data collected on a satellite can be downlinked to the earth when it has contact with a ground station. The data is typically compressed and downlinked for further processing on Earth. This feed-forward model has the downside of using bandwidth to downlink all data regardless of its value. Unchanged areas may not provide valuable information and thus only a report of what changed or the images containing changes may be of value. Bandwidth can be relatively limited while data volumes collected by multispectral, hyperspectral, and SAR sensors can be large [2], creating a bottleneck or costly operations.

The feed-forward model also induces a latency from time of data collection to when it is all available and processed on ground. In the case of natural disaster monitoring and other time-critical tasks, near real-time relay of valuable information is of importance. However, the upside of the feed-forward model is that all data is available for storage and processing on high-performance computers on Earth.

2.2.1 On-board computing

Advanced on-board computing has been identified as an important factor for applications in autonomous rendezvous, low latency science missions, exo-planetary avionics and downlink bandwidth limited missions where the raw sensor data volume is greater than available downlink bandwidth [2]. More on-board data processing is needed to handle the issues of latency and bandwidth bottlenecks [26], as illustrated in Figure 2.2.1. The development of on-board AI is considered as an active area of

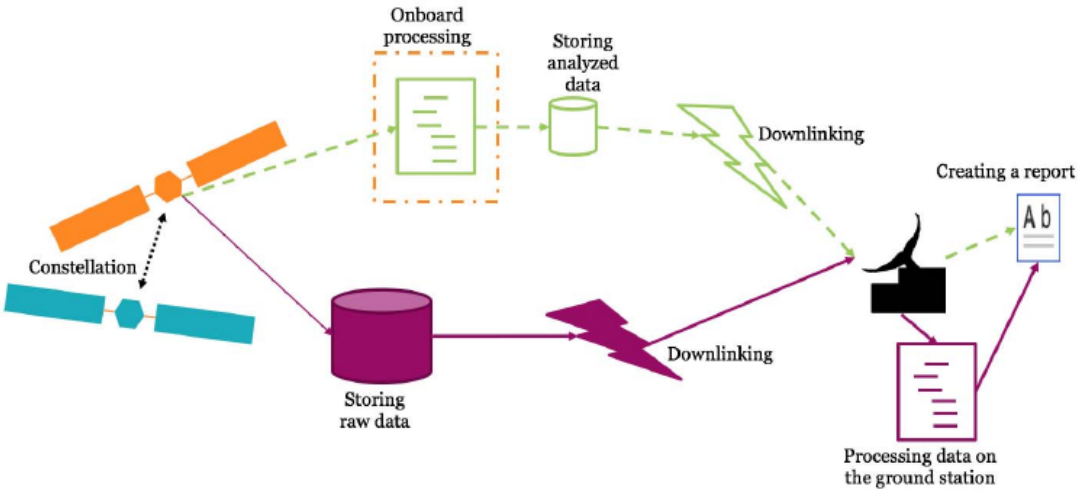


Figure 2.2.1: Downlinking and processing diagram from [26]

research for the ESA Φ-Lab [15].

On-board data processing needs to function within the Size, Weight and Power (SWaP) limits that satellites have, in addition to being reliable in a radiation environment [26]. To meet the demands of sensor data processing especially in SmallSats is a major challenge where SWaP and cost constraints are tighter.

Radiation hardened OBCs have been lagging behind on compute power compared to commercial ground computers [9]. The use of lower-cost high-performing Commercial-off-the-shelf (COTS) components for small satellites is becoming more frequent in hybrid OBCs that utilize multiple kinds of computing technologies such as radiation hardened and commercial components for speed and reliability. Using commercial components, performance requirements for energy, speed and storage can more easily be met [9].

Using commercial processors is still a major challenge due to radiation effects [9]. Radiation tolerant commercial OBCs are however emerging for the CubeSat market. Heterogeneous or hybrid systems utilizing any combination of Central Processing Unit (CPU), Graphical Processing Unit (GPU), Field Programmable Gate Array (FPGA) and other specialized hardware open up for intelligent on-board processing [2, 9]. For example, image processing and computer vision tasks could be delegated to the GPU for hardware acceleration. Even more specialized components for computer vision are emerging, such as Visual Processing Unit (VPU). GPUs have rarely been used on

satellite computers due to high power requirements, although this is changing as they are integrated on System on Chip (SoC) systems such as Unibap's iX5.

2.2.2 High-performance on-board computers

Table 2.2.1 lists a sample of OBCs with hybrid architecture or COTS deemed capable of potentially running CD algorithms on-board in addition to standard command and data handling. Some hardware models were not included due to lacking technical specifications. The list is not exhaustive of every relevant on-board computer on the market.

The devices vary in components, capability, and architecture. It is therefore difficult to identify a typical satellite computer. However, knowing that there are SoC systems with multi-core CPUs and more notably GPUs indicate the possibility of running ML inference engines on the devices efficiently. FPGAs are common on OBCs and could potentially be used for ML , although it is not considered in this work. Notably, Unibap and Ubotica provide Intel Movidius Myriad VPU support which could greatly enhance performance for visual ML tasks such as CD.

Note that reliability, i.e. radiation tolerance, has not been considered in the comparison as it does not affect the performance directly. In practice, modern COTS processors often perform 10 to 100 times better than radiation hardened processors [2]. Reliability requirements depend on the particular mission.

2.3 Change Detection using ML

Change detection is the process of finding changes of interest in an area between two points in time. Examples of changes to track are the use of farmland, water bodies, buildings and roads for use in applications of agriculture, environmental monitoring, and urban planning, respectively [10, 22].

The CD process produces a change map which provides information about what changes of interest may have occurred between the images and where. Change maps can use any of the following mappings: [22]

- Binary. Change or no change.
- One-class

- From-to. From one class to another.
- Instance. Changes to identified objects.

There are numerous classical models for change detection. These models typically rely on static numerical methods to analyze differences in the images [10]. While these are useful, this thesis focuses on ML based methods.

In recent years, Deep Learning (DL) based computer vision has developed significantly and has successfully been applied to Remote Sensing (RS) images. CD in particular has significant research suggesting ML methods to be more accurate than classical methods [10]. By using DL over classical methods, features of images can be learned instead of handcrafted. This learning can lead to the capturing of high-level features as well as more subtle variations to potentially surpass the accuracy of classical methods. ML methods have shown promising results and improve the feature extraction capability of classical methods in numerous reported cases [22].

Some types of models that have been researched for CD are CNN, RNN, GAN, Auto Encoder (AE) and Deep Belief Network (DBN). CNNs and RNNs can be used in a supervised fashion, in which case a carefully labeled dataset serves as examples of what constitutes a change and not, i.e. a set of correct change maps. However, labeled datasets for CD are typically scarce or small [15, 22].

With limited labeled data in mind, effective unsupervised methods are desirable. These methods learn patterns in data without labels. Latent features are learned by e.g. an AE and change maps can be produced from the latent space by a cluster algorithm [22]. Weakly supervised and semi-supervised methods lie in between fully supervised or unsupervised. An example of such attempt for CD is automatic labeling of pixels as certainly changed/unchanged or uncertain, where only the certain pixels are used as labels automatically [4].

An ML-based CD process generally includes following steps [31].

1. Homogenization of data
2. Training set generation
3. Model training
4. Model serving (inference)

The interest of this thesis is doing model serving on an OBC instead of on the ground.

The optimal choice of basic unit for feature extraction depends on the change detection application. Pixel-based CD is detailed but inevitably noisy. Patch-based methods can be appropriate for scene changes where details are overlooked. Object-based change detection can achieve a compromise between the former two but can be sensitive to scale and target characteristics [31].

2.3.1 Convolutional Neural Networks (CNN)

CNNs are neural networks that leverage convolutional layers to extract local spatial features in the input. Accordingly, they have been successfully applied to computer vision tasks, including CD. Kernels convolve over the input to produce a feature map highlighting where the feature was found. As the convolutional layers stack with pooling layers in between, high-level features are learned. With pooling, memory and sensitivity to feature location are reduced as the input is downsampled. In CD, CNNs often play the important role of feature extractor (encoder).

2.3.2 Siamese CNN

Siamese CNNs are dual channel networks with shared weights that take different images as input. As seen in Figure 2.3.1, the feature vector outputs of the dual networks are combined by some process to a single change map output. The role of siamese CNNs is to extract high level features, rather than pixels, to have more robust and nuanced information on which to compare changes.

In practice, because all weights are shared, the same CNN network can execute twice in sequence before being processed with change analysis. There are also variants such as pseudo siamese and asymmetric siamese variants where weights are only partially shared [29]. In this case, the dual networks are invoked and stored differently.

Transfer learning can be applied to siamese CNNs to reduce the number of training samples needed for adequate feature learning [22]. A pre-trained network may have learned relevant deep features to leverage in training on a specific dataset. The network can be fine-tuned by further training on domain or task specific images.

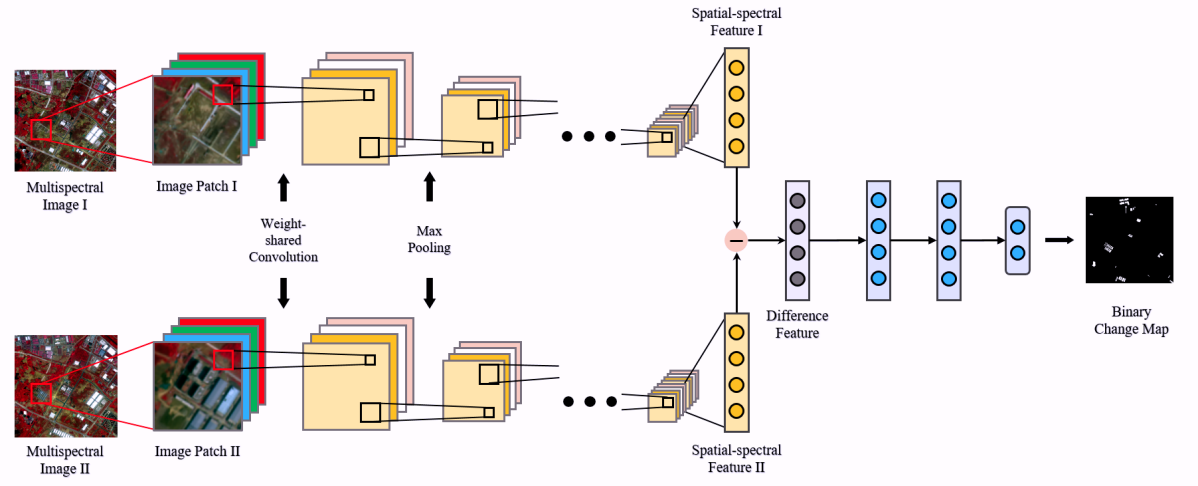


Figure 2.3.1: Example of siamese CNN structure from [5] for directly inferring binary change maps

2.3.3 Semantic Segmentation

The goal of segmentation is to classify each pixel of an image to localize the objects found, as seen in Figure 2.3.2. As such, it is useful in CD where localization of changes to certain objects such as buildings are of interest.

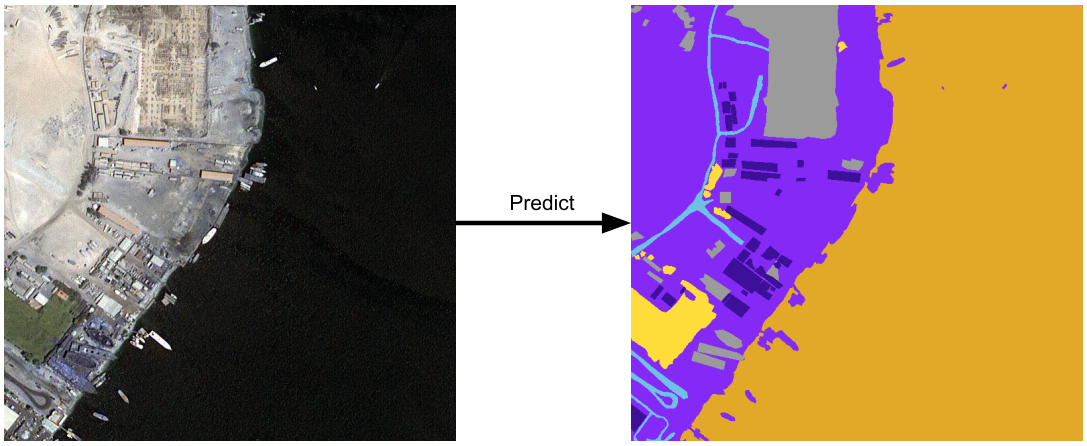


Figure 2.3.2: Semantic segmentation of a satellite image of an urban area [12]

Semantic segmentation networks such as Fully Convolutional Network (FCN) [21], SegNet [1] and U-Net [17] work on the principle of an encoder-decoder structure. The encoder downsamples the image through convolutions to find *what* is in the image and produces feature maps containing information about the features found. The decoder then upsamples the feature maps by transpose convolutions to determine *where* the found objects are located and finally provides a classification for each pixel. FCNs and variants like U-Net do not require the input images to be of a specific size contrary to CNN networks containing fully connected layers [21].

It is common for supervised CD methods to rely on segmentation networks. Examples used in this work are [4, 6, 14]. Segmentation can be used directly in post-classification as described below in section 2.3.4. It can also be used in other architectures where two images are combined to segment changes directly.

2.3.4 Post classification

The post classification approach to CD uses a segmentation network to classify each pixel in a bi-temporal pair of images. The classification maps are then compared to find changes in the classification, hence the name *post*-classification. The accuracy of the change map then depends on the accuracy of the classifier. It does however accumulate errors in the classification process [31]. Post classification can be applied to

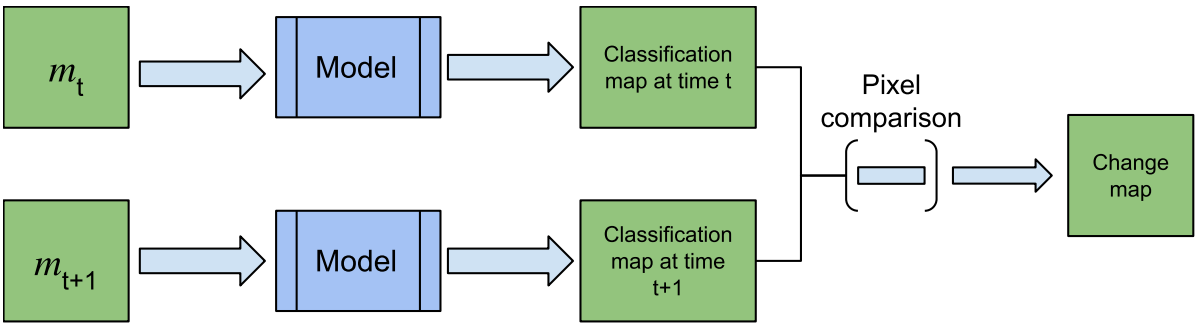


Figure 2.3.3: Post-classification structure for change detection in image pair $\{m_t, m_{t+1}\}$

binary classification as object or no object of interest, or multi-class classification used in e.g. land cover. The structure has proven suitable for urban changes where objects like buildings and roads are fairly well defined [31]. ML methods have successfully been applied to land-cover classification in multiple studies [22] enabling it for CD of land-cover.

Post-classification is known to be robust in varying illumination, attitude and season conditions [22]. Because only the classification maps are compared, storage requirements depend on the size of the classification map which in turn is subject to dense representation through bitmaps or sparse matrices for example.

2.4 Related work

AI methods for CD have been reviewed by Shi et al. [22]. The authors concluded the following recommendations for model selection depending on factors not including on-board data processing:

- When image data is heterogeneous, for example, SAR and optical images, pseudo-siamese CNN models can account for differences in data by shared only some weights.
- Post-classification methods can be used when the change map needs to encode a from-to mapping in addition to a binary changed/unchanged classification. They first classify pixels or patches in the images into a set of classes and then compare the difference in classification to produce the change map.
- For long-term sequences of images, models can benefit from having an RNN as part to capture sequence patterns.
- In the case of insufficient annotated training samples, transfer learning, AE or GAN, models can be trained without annotated samples.
- If sufficient labeled samples are available, however, (siamese) CNN models perform well.

Challenges for change detection with AI were identified by the authors to be the following:

- **Heterogeneous data** The myriad of sensor types available requires methods to be able to compare data of different types or do sensor fusion.
- **Unsupervised learning** The lack of large labeled datasets require unsupervised methods. These methods however need more improvement in accuracy. Weakly supervised or semi-supervised methods are promising ways to deal with lacking labels.
- **Reliability** AI models are not guaranteed to work well. They can be seen as black-boxes that limit understanding and reliability.

This work uses homogeneous data, a moderate size labeled dataset and is not concerned with maximizing accuracy. As such, the main challenge tackled in this thesis is the on-board processing system.

2.4.1 Urban change detection

The SpaceNet *Multi-Temporal Urban Development Challenge* in 2020 called for solutions to building tracking and change detection in time-series of satellite images. It supplies the largest labeled multi-temporal building dataset to date. In contrast to most research on building CD [3, 4], the dataset is at 4 m per pixel resolution which is lower than the VHR images often used. Small pixel counts for each building pose a challenge for robust CD.

A baseline algorithm for the SpaceNet7 challenge is outlined in [8]. It uses a segmentation model to classify building pixels and then convert those to polygons describing the shapes of the buildings found. Each polygon is assigned an identifier and tracked over subsequent images in the time-series. The segmentation model is a U-Net+VGG16 architecture. The winning solutions all used segmentation models, sometimes in a siamese configuration [28]. One other relevant technique used in top solutions was upsampling of images 3-4x before feeding to the network to better handle the low resolution. Most top performing solutions used ensemble methods which significantly slowed down inference. A notable exception was the winning solution.

All solutions to the challenge differ from the goal of this work in that they track longer time-series rather than bi-temporal image pairs and identify changes in each building individually as compared to any changes in buildings.

2.4.2 On-board Machine Learning

Deep learning using TensorFlow was shown by Bruhn et al. [2] to be applied efficiently on an AMD SoC with GPUs to do Earth observation tasks in orbit. Further, they claim near real-time on-board processing and training. In contrast from this work, their newer chip supported ROCm and extended the GPU with an Intel Movidius Myriad X accelerator. The ML workload was time-series anomaly detection, which is a computer vision task although different from CD.

Performance of vision tasks using ML on a CPU+GPU SoC for an OBC similar to Unibap's experiment system have been researched by Tsog et al. [26]. More specifically, using a Heterogeneous System Architecture (HSA) compliant SoC they concluded that using a GPU can be two orders of magnitude faster than CPU for

intelligent image processing. Moreover, the energy usage was an order of magnitude less than that of CPU. The authors conclude that GPU can be a ”highly potential candidate in on-board computer processing”.

2.4.3 On-board Change Detection

A method for on-orbit CD using ML has been proposed by Yang et al. [30]. The method is designed for SAR data, which excludes direct use of the model for experiments in this thesis. However, design choices in their proposed Extreme Self-paced Learning Machine are relevant to any on-board processing. The method is ”accurate and robust in detection, parameter free and low complexity” to suit on-board processing.

2.4.4 Patch-based change detection

A siamese CNN architecture for change detection in patches has been implemented by Rahman et al. [16]. In their model, a patch of 80×80 pixels is classified as changed or not, where a change is defined as having a helicopter enter or exit the patch between two points in time. Activations from one or more convolutional blocks in a VGG16 architecture are concatenated and connected to a decision network. The decision network has one hidden layer and a single output for change or no change decision. Another approach attempted was to compute the Euclidean distance of the block vectors between points in time and decide using a threshold value.

2.4.5 MobileNet

MobileNetV2 [18] is a neural network architecture created in 2018 specifically designed for mobile and resource constrained applications. At the time of publication, the network was SOTA for mobile computer vision models. It works by reducing the memory footprint and number of operations required while keeping most of its accuracy.

Table 2.2.1: Sample of high performance OBCs

Manufacturer	Model	CPU			GPU	VPU	FPGA	RAM (GB)	Storage (GB)
		Cores	Freq	Arch					
Unibap	iX5	4	1.2 GHz	x86	Radeon R3E	Myriad 2	SmartFusion2	2	64+240
Unibap	iX10	4	2.0 Ghz	x86	Radeon Vega	Myriad X	PolarFire	8	<240
Ruag Space	Lynx	4		ARM	-	-	Yes	8	64
Ubotica	UB0100	-	-	-	-	Myriad 2	-	4	SD card
ibeos	EDGE	4	2.0 GHz	ARM	192x CUDA	-	-	2	4
Innoflight	CFC-500	4			192x CUDA	-	Ultrascalable+	16	16
Hyperion	CP400.85	1	0.5 GHz	ARM	-	-	-	0.5	0.5-7.5
Moog	Bandera	1	>1GHz		-	-	Yes	>24	>48
Moog	G-series SBC	1	1.2 GHz	x86	Yes	-	KU060	2	64
SEAKR	Athena-3	1	1 GHz		-	-	-	1	1-4

Data is of varying quality or existence due varying levels of public information. CPU frequency is not a direct measure of compute power, but an indication.

Chapter 3

Method and experiments

This chapter describes and motivates the method of evaluating technical feasibility of the models. The proposed use case, selected models, novel model design, dataset and experiment design is described. Finally, various implementation details are outlined for reproducing of results and additional details.

The proposed use case of the CD methods is illustrated in Figure 3.0.1. The satellite takes an image m_t of an area a at time t and stores necessary data d_t on the satellite after processing. When revisiting the area a at time $t + 1$, the satellite takes another image m_{t+1} and together with stored data d_t produces a change map which determines what data should be downlinked and deleted from satellite storage. Furthermore d_{t+1} is stored and the process repeats. The system leverages on-board processing capabilities to potentially reduce bandwidth and latency in ground communication as per Figure 2.2.1.

During experiments, image pairs m_t and m_{t+1} were loaded from file and processed without a time delay in between. Storage buildup due to revisit times was instead modeled theoretically as it was entirely predictable from network architectures. In doing so, experiments could be run far quicker while accurately modeling the proposed operational system. The computer was situated on the ground but behaves no differently from when in space except for being shielded entirely from radiation-induced errors. Radiation effects are beyond the scope of this work.

Evaluating feasibility is part quantitative and part qualitative. Experiments provide quantitative data on runtime behavior. The collected data forms the basis for evaluating the feasibility of the system where theoretical constraints and hypothetical

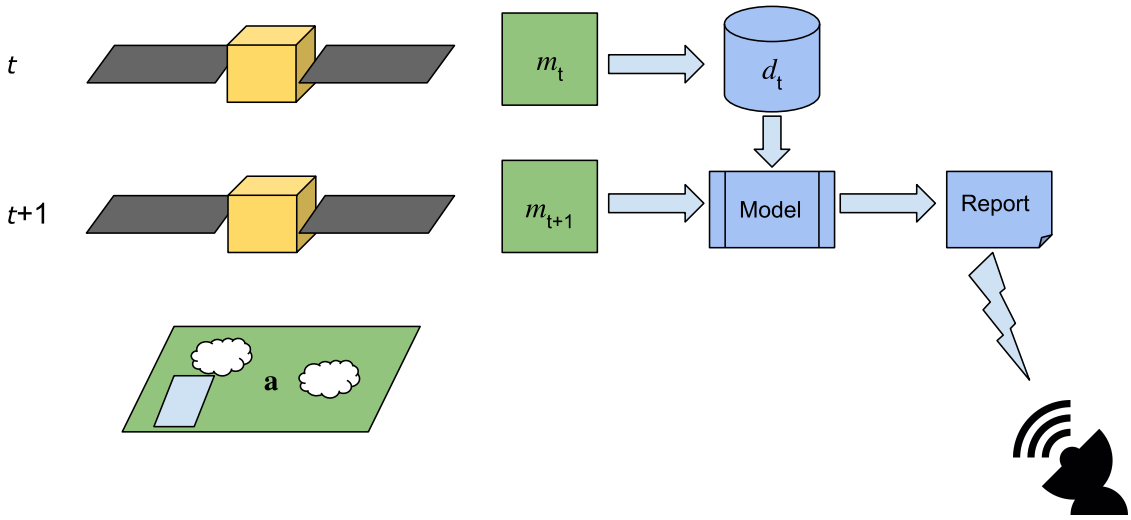


Figure 3.0.1: Proposed use case

scenarios are introduced, see section 3.6.

Desired characteristics of on-board CD methods are short computation times, low storage requirements between images, low energy consumption, and high accuracy. It was expected to encounter trade-offs between these traits. Hard constraints are determined on a case-by-case basis. Consequently, the quantitative runtime experiments on a single machine are not sufficient to fully answer the question in the most general sense. Quantitative analysis is supplemented with an evaluation of technical feasibility in relation to modeled use case scenarios, other hardware, software, and satellite configurations.

Model accuracy is considered in so far as being sufficient or not for practical application. Large variations in accuracy are taken into account, but accuracy is secondary to computation time and storage. Accuracy is described primarily by metrics precision, recall, and F1 score to account for dataset imbalances.

The work takes two high-level approaches to change detection; pixel-level and patch-level change classification. In the former, each pixel is classified as changed or not in a change map whereas in the latter the patch is classified as containing changes or not. The pixel-level approach is further split into direct classification and post-classification approaches.

The dataset used is SpaceNet 7 with preprocessing as described in section 3.3. It was

chosen primarily due to its relatively large number of labeled samples.

3.1 Pixel-level change detection models

Available SOTA CD methods targeted at optical satellite images were selected from the literature. While the optimal choice of method can depend on the changes to detect, all of the selected models have been tested directly on building changes or are applicable to this use case. Only papers with open source implementations were used for practical reasons.

Pixel classification models were used in one of two ways; direct classification and post-classification. Direct classification refers to all models that take two images as input and ”directly” produce a change map, i.e. siamese and early fusion networks. Post-classification uses segmentation models to semantically classify pixels of each image and then compare such classifications to find changes.

An overview of selected pixel-level models is found in Table 3.1.1.

Table 3.1.1: Pixel level change detection models compared

Approach	Name	Publication Title
Direct classification	FCSD, FCSC, FCEF	Fully Convolutional Siamese Networks for Change Detection [6]
	DSMSCN	Change Detection in Multi-temporal VHR Images Based on Deep Siamese Multi-scale Convolutional Neural Networks [4]
Post classification	MobileUnet	Semi-novel model, see section 3.1.4
	xdxd	The SpaceNet Multi-Temporal Urban Development Challenge [28]

TensorFlow models were exclusively selected in order that they can be converted to TensorFlow Lite (TFLite) models that are optimized for embedded devices such as the Qseven e20xx compute module used in the experiments. Furthermore, using the same framework for all models makes runtime comparison more accurate by controlling for potential variations in framework performance.

3.1.1 Post-classification

Post-classification models follow the general architecture as described in section 2.3.4. Images were segmented one at a time using the semantic segmentation model to classify building pixels. The resulting binary segmentation maps were then compared pair-wise for changes in classification using the same method as when creating change maps, described in section 3.3.1.

Predicted change map patches from segmentation models were stitched together to 1024×1024 images during inference.

3.1.2 FCSD, FCSC and FCEF

The fully convolutional family of change detection models Fully Convolutional Siamese Difference (FCSD), Fully Convolutional Siamese Concatenated (FCSC) and Fully Convolutional Early Fusion (FCEF) [6] have successfully been used for urban change detection in data with lower resolution than SpaceNet 7 (SN7). The models were trained directly using pairs of 512×512 images with corresponding change map as label. No pre-trained weights were used. Onera Satellite Change Detection (OSCD) dataset was used in addition to the SpaceNet 7 dataset for comparison to results in [6].

3.1.3 DSMSCN

DSMSCN uses multi-scale convolutions. DSMSCN is according the authors Chen et al. more accurate while being 46.9% and 38.4% smaller than FCSC and FCSD, respectively [4]. Furthermore, they claim an inference time under 0.1 seconds per image for DSMSCN. Therefore it was an interesting model. The network was trained the same way as the fully convolutional family of networks in section 3.1.2.

3.1.4 MobileUnet

MobileNetV2 is known to be efficient on edge devices and have good accuracy [18]. MobileNetV2 was used as an encoder to create a UNet architecture model for semantic segmentation, referred to here as *MobileUnet*. It was designed to evaluate a post-classification approach using mobile tailored TFLite compatible models.

The MobileUnet model was trained on SpaceNet 7 image-mask patch pairs of size 512×512. Pre-trained weights for 224×224 input size MobileNetV2 was used to leverage transfer learning.

3.1.5 SpaceNet 7 baseline - xdxd

The SpaceNet-7 challenge provides a baseline segmentation model called *xdxd* [28]. It is a PyTorch model, which disqualifies it for inference benchmark experiments in this work. However, it is pre-trained and meant to serve as a baseline for SpaceNet 7 dataset segmentation which makes it useful as a reference point to other segmentation models used in this work.

The pre-trained PyTorch model *xdxd* was used without fine-tuning in a post-classification structure for reference. The *xdxd* model is missing benchmarks due to being implemented in PyTorch.

3.2 Patch classification method

The patch classification method classifies the whole input patch as changed or not. This method is fundamentally different from pixel classification methods that preserve spatial information by classifying each pixel. Patch classification more closely resembles image classification than segmentation. Spatial resolution is proportional to the size of patches used as input. We believe that a patch classification model can achieve the goal of reducing bandwidth by only downlinking patches that have relevant changes, although not to the same extent as pixel-level methods.

A novel model called MobileGoNogo was designed for patch classification.

3.2.1 MobileGoNogo model

Based on siamese networks described in [16], a novel binary classification model called MobileGoNogo was designed to classify an image patch as containing changes or not.

The model architecture can be seen in Figure 3.2.1. It consists of siamese MobileNetV2 classifiers with shared weights and a decision network. The top classification layer of MobileNetV2 is left out and a global average pooling layer is applied to the last

block. The feature vectors produced by MobileNetV2 in this configuration are of size 1280. The pair of feature vectors are concatenated and used as input to a hidden fully connected layer which ultimately connects to a single output neuron with sigmoid activation for binary classification of change or no change.

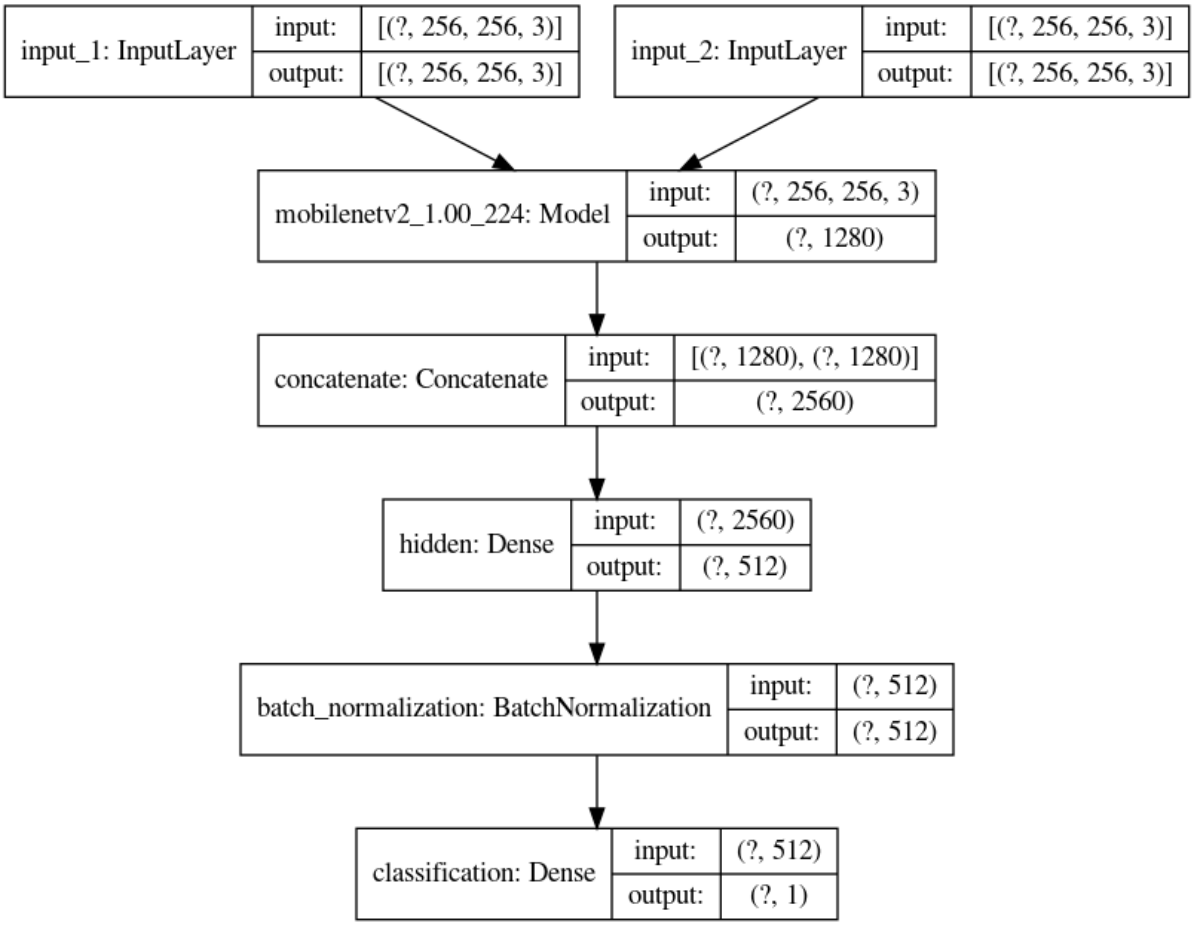


Figure 3.2.1: MobileGoNogo model architecture

The main idea for this type of network is that the feature vector is the only data d_t that needs to be stored after capturing an image m_t and that relevant differences in feature vectors can be learned by a decision network. The hidden/intermediate decision layer was set to size 512 after initial experimentation showed a layer size of 1024 yielding similar results as 512.

The model was trained with and without the 2x upscaling used with segmentation models. Because upscaling was not previously proven effective on a classifier network it was not assumed to be effective as it was with the pixel-level models. Patch sizes were 256 or 512 pixels wide. Without upscaling only 256-pixel patches were used.

Freezing encoder weights and subtracting feature vectors rather than concatenating

was attempted but showed worse results. MobileNetV2 pre-trained weights for 224×224 images were used for the siamese encoders. Binary Cross Entropy (BCE) was used as a loss function.

3.3 Dataset

The labeled dataset used was *SpaceNet 7: Multi-Temporal Urban Development Challenge* [8]. It covers 100 locations with a medium resolution at 4 m per pixel. For reference, Sentinel-2 has 10 m at minimum. The dataset is the largest publicly available dataset at the time of writing to the best of our knowledge, containing 2389 images with $\approx 11 \times 10^6$ annotated buildings. Serving as a basis for a NeurIPS 2020 challenge [13] it is used for other current-day research. Specifications of the dataset can be found in Table 3.3.1.

Table 3.3.1: SpaceNet 7 dataset specifications

Num AOIs	101
Num Observations	2389
Num Buildings	11,080,000
Total Observed Area (km2)	41,000
Mean Buildings per Observation	4,700
Mean Building Area (m2)	190
Mean ground sample distance (resolution) (m)	4.0

The OSCD dataset was used as a secondary dataset for reference. Direct classification models used in this work have accuracy results reported on the OSCD dataset in [4, 6].

3.3.1 Change maps

The SN7 dataset included annotated masks of *building* or *no building* pixels. It did not include change maps between pairs of images. Change maps were created by comparing each labeled pixel (1 or 0) of the masks do find differences according to formula $|p_t - p_{t+1}|$ where p is a pixel at time t. Unusable parts of images were masked as containing no buildings or changes.

Due to low change between each month, time intervals of 3, 6, and 12 months between images were chosen to create less imbalance in labels. Using intervals reduces the number of change maps by the number of months in the interval for each series of 24 images. As seen in Table 3.3.2, SpaceNet 7 is extremely imbalanced which makes learning harder.

Longer time intervals naturally have more changes but not necessarily changes that only occur during long time frames. A building can appear from one month to the next and a longer time interval may simply increase the number of such changes. Therefore, a long time interval can be used for training even though the model is intended to be used on shorter intervals.

Change maps were provided in the OSCD dataset. The only preprocessing on this dataset was patching to 256×256 half-overlapping patches.

Table 3.3.2: Ratio of unchanged to changed pixels in images. Ratio for total pixels in datasets and mean ratio per image in respective dataset is given.

Dataset	Ratio (tot)	Ratio (mean)
SN7 (3 month)	698	2449
SN7 (6 month)	145	977
SN7 (12 month)	110	380
OSCD	43	80

Only one binary label is needed for each image in patch classification. The label (change/no change) was derived by a threshold of changed pixels in the change maps created. Threshold values 1, 4, and 8 were tested for the number of changed pixels required to classify the whole patch as changed. For patches of 256×256 pixels, the dataset was reasonably balanced for any of the thresholds found in Table 3.3.3.

Table 3.3.3: Binary label distribution for thresholds of changed pixel count on 256×256 pixel patches

Threshold	Part positive
1	61.3%
4	60.3%
8	56.9%

3.3.2 Scaling and patching

Raw image inputs were upscaled by a factor of 2 in width and height such that each pixel became 4. It was proven to be an effective method to increase accuracy in all winners of the SpaceNet 7 challenge [28] and was therefore used with all pixel-level models. With 4m^2 per pixel, some buildings only cover a few pixels in the image making it difficult for the CNN to accurately learn the patterns.

The images had an original size of 1024×1024 pixels. After upscaling, the size was 2048×2048. Non-overlapping patches of 512×512 were extracted to keep networks reasonably lightweight for running on resource-constrained hardware. The upscaling and patch extraction process was performed on the fly during training. Patches were shuffled and then batched.

Images were converted from GeoTIFF format to PNG using ImageMagick for easier loading to training. Because TIFF and PNG are lossless formats, the conversion does not affect image quality. Some metadata is lost in the process but it does not affect training.

3.3.3 Augmentation

The following augmentation of the dataset was performed during training to artificially increase the dataset size:

- Flip left-right
- Flip up-down
- Rotate random multiple of 90 deg

Each of the augmentations had a 50% probability of being applied during training to an input-output pair, i.e. a 512×512 patch pair along with its corresponding mask. The random augmentation, as well as shuffling of batches, was introduced for each epoch.

3.3.4 Training and test data split

The test partition of the dataset did not include labels. Therefore, a number of images were selected from the training dataset. Out of the 60 Area of Interests (AOIs) in the training set, the 15 listed in Table 3.3.4 were extracted as test set for evaluation of

Figure 3.3.1: Sample of upscaled input image patches and corresponding change map

models leaving 45 AOIs for training. The split was 25% and 75% for test and training sets, respectively. The choice of the first 10 AOIs was taken from the winning solution of the SpaceNet 7 challenge and the other 5 were selected at random.

Table 3.3.4: Test AOIs

L15-0387E-1276N_1549_3087_13	L15-1276E-1107N_5105_3761_13
L15-0566E-1185N_2265_3451_13	L15-1438E-1134N_5753_3655_13
L15-0632E-0892N_2528_4620_13	L15-1615E-1206N_6460_3366_13
L15-1015E-1062N_4061_3941_13	L15-1690E-1211N_6763_3346_13
L15-1200E-0847N_4802_4803_13	L15-1848E-0793N_7394_5018_13
L15-1481E-1119N_5927_3715_13	L15-1203E-1203N_4815_3378_13
L15-1389E-1284N_5557_3054_13	L15-0506E-1204N_2027_3374_13
L15-0577E-1243N_2309_3217_13	

3.4 Prototype computer

Unibap's Qseven compute module e20xx used on the prototype machine is the same model used in Unibap's iX5 satellite computer. It uses the AMD GX-412HC SoC and Microsemi SmartFusion2 FPGA. The SoC chip is targeted at embedded applications with low Thermal Design Power (TDP) requirements which allows it to fit the SWaP constraints of small- and nanosatellites. The machine used in the experiments had a 120 GB SSD, which is within the limits of 240 GB SSD storage that the iX5 is capable of. Experiments were run such that only the CPU, GPU and RAM significantly affect the results. Therefore, the results were expected to be equivalent to the iX5100.

In comparison to other hardware in Table 2.2.1 it is on the higher end in terms of performance but represents a realistic choice for current and future missions

nonetheless.

The prototype computer was expected to limit CD capabilities by not having enough RAM for larger networks and not be able to efficiently run custom layer operations that are not supported in the TFLite framework. Some precision was expected to be lost through quantization.

Like the iX5 OBC, the prototype computer had an optional Movidius VPU accelerator. It was not used since it is so rare among satellite computers in Table 2.2.1 to have such accelerators. It is however a technology with some flight heritage on ϕ-Sat 1 and potential for efficient AI inference.

3.5 Inference benchmarks

After training, the models were converted to TensorFlow Lite models using the official converter with varying optimization settings. Inference time was observed on the prototype machine where TFLite models were deployed. All models were run 30 times after warmup and compute time was observed. Mean and Standard Deviation (SD) were then computed.

The XNNPACK delegate was used to speed up CPU floating-point inference for TFLite models without quantization. The models were tested without XNNPACK as well to verify its effect. Quantized models were tested using optimization flag

```
tf.lite.Optimize.DEFAULT
```

The setting enables 8-bit quantization for weights but not for activations. It is generally expected to achieve 2-3x speedup with some loss in accuracy on CPUs.

At first, 16-bit quantization was attempted to get better performance on the GPU. The GPU delegate failed to utilize this. TFLite GPU also failed to load OpenCL and used OpenGL as a fallback on all models except MobileGoNogo.

Memory usage statistics were collected using the official TFLite benchmarking tool [25]. Because the tool itself uses memory, the numbers are only approximations.

3.6 Use case scenarios

Two different scenarios based on currently operational satellite configurations were modeled; a traditional monolith satellite and a large constellation of small satellites.

Scenario 1 One satellite performs the change detection. Its sun-synchronous orbit yields a revisit time of 10 days at the equator. No inter-satellite communication is required. This is based on configuration of Sentinel-2 [20].

Scenario 2 A large constellation of CubeSats in sun-synchronous orbit yielding daily revisit time. Data sharing capability between satellites is assumed to be in place. This is based on Planet's PlanetScope constellation [19].

3.7 Implementation details

Repositories for existing compared models were forked and combined in a parent repository containing the data loader and training scripts. MobileUnet and MobileGoNogo were created from scratch. The GitHub repo can be found at `https://github.com/solivero/onboard_change_detection_thesis`

Existing models were ported from TensorFlow version 1 to version 2. Keras code was also ported to utilize TensorFlow (TF) 2 built-in Keras modules. Code for models DSMSCN, FCSD, FCSC and FCEF was taken from GitHub repository `https://github.com/I-Hope-Peace/DSMSCN`. The FC family of models have an official implementation in PyTorch. However, the DSMSCN repo provided TF code for their models as well for reference.

As training was observed to be unstable for DSMSCN and FCSD models, disabled batch normalization layers found in the model code were enabled. All models used the same data loader with augmentation and preprocessing as described in 3.3.3.

3.7.1 Hardware

Unibap e20xx compute module - Development environment for prototype

Operating systems

OS: Ubuntu 18.04 bionic

Kernel: x86_64 Linux 5.4.52-050452-generic

Shell: sh

Hardware specs

Chip: AMD GX-412HC SoC

CPU: Jaguar+ - 4 core @ 1.2GHz (1.6 turbo) [40.0°C]

L2 Cache: 2MB

GPU: Radeon R3E Graphics - 2CU @ 267 Mhz (350 max)

RAM: 1716MiB DDR3 EDAC

Storage: 119 GiB, SATA SSD

TDP: 7-15W

Supported technologies

DirectX® 11.2

OpenCL® 1.2

OpenGL 4.3

The prototype machine Operating System (OS) was Ubuntu 18.04 (Linux) with SpaceCloud [27] tools and drivers. It supported common ML frameworks such as TensorFlow. TFLite in particular has optimizations for embedded devices like the SoC used and supports the GPU. It was therefore preferred. TFLite was configured to use OpenCL over OpenGL to maximize performance on GPU when possible. For CPU it was compiled with XNNPACK for 32-bit float optimizations on x86 architecture. It was benchmarked with *clpeak* to 3.6 TFLOPS. Technologies not supported but expected in the upcoming iX10 are ROCm with OpenCL 2.0 which could improve performance on GPU and CPU. The GX-412HC on the iX5100 has been benchmarked at 87 GFLOPS for OpenCL GPU and scored 5,842.98 at CoreMark v1.0 per CPU core [23].

Training was performed on a compute server with the following hardware specifications. Only one GPU was used at a time for any particular model training session.

- GPU: 4x Nvidia GTX 1080 Ti (11GB)

- CPU: AMD Ryzen Threadripper 1920X 12-Core Processor 48 threads @ 3.5GHz
- RAM: 126GB

3.7.2 Hyperparameters

All models used the Adam optimizer [11] with the learning rate set to 2×10^{-4}. This is consistent with the original code.

Early stopping was used during training to avoid overfitting. The patience parameter was set to 10 epochs, monitoring validation loss, such that training is stopped when validation loss has not increased during the previous 10 epochs. As a result, the models had a varying number of epochs of training depending on their training behavior. The maximum number of epochs were 50.

3.7.3 Loss functions

Loss functions weighted binary cross-entropy, dice loss [24], and combinations of them were tried. Class weights were settled on 0.1 and 0.9 for no-change and change, respectively. Automatic weighting for each batch was tried but showed poor training convergence. A combined weighted BCE and Dice loss

$$l_{BCE} + 0.5 \cdot l_{Dice}$$

was used for the final reporting of all results.

We believe that auto weighted and pure Dice loss do not perform well on SpaceNet 7 because many training examples do not contain a single positive label. It causes both losses to suffer from division by zero leading to bad gradients.

3.7.4 Inference benchmarks

Measurement of inference latency was performed as following:

```
float* input = interpreter->typed_input_tensor<float>(0);
float* output = interpreter->typed_output_tensor<float>(0);

// Warmup to make sure cache are warm and any JIT compiling
// is in the way has run/tuned
```

```
interpreter->Invoke();

// run single invocation and measure time
auto t1 = std::chrono::high_resolution_clock::now();
interpreter->Invoke();
auto t2 = std::chrono::high_resolution_clock::now();
```

TFLite benchmarking tool was also used to verify results.

Chapter 4

Results

Results for model accuracy, complexity, inference time, RAM, storage requirements, and use case modeling are presented. Finally, a summary of the different results is presented to assess the overall technical feasibility of each model.

4.1 Inference benchmarks

Mean and SD from inference benchmarks can be found in Table 4.1.1 as well as in Figure 4.1.1. The novel MobileGoNogo model was the fastest. Compared to MobileUnet, the next fastest model, MobileGoNogo was faster by a factor of 7.7 and 10.8 for single-core and quad-core, respectively. Amongst pixel-level methods, MobileUnet was the fastest. Unlike the other models, however, MobileUnet only processes one input image.

The XNNPACK delegate showed an order of magnitude faster inference times in most models and was therefore used in all presented CPU benchmarks. The reason for the drastic speedup is that XNNPACK is optimized for x86 architecture (utilizing AVX instructions) while the old TFLite backend was targeted at ARM. Input images were 512×512 pixels in all cases.

No speedup was found when using 8 bit integer quantized models. The reason is that XNNPACK is specifically optimized for floating-point operations and not using XNNPACK is not optimized for x86, leaving no opportunity for quantization to work efficiently on the particular hardware. Quantization did however reduce the model size as expected. Absolute model size reduction ranged between 0.5 MB to 4 MB.

Utilizing quad-core reduced compute time significantly. Average ratio of quad-core to single-core performance was 2.87 across models, indicating successful parallelization of models. Standard deviation of compute time was an order of magnitude higher on the multi core setup for each of the models FCSD, FCSC and MobileUnet. The other models did not see a significant change in variance. Consequently, the variance between models also increased in quad-core experiments.

Memory usage ranged from 191 MB to 556 MB corresponding to 9.3% to 27.6% memory usage, respectively, on the prototype machine.

GPU performance was lower than expected. Even without any quantization, the GPU was significantly slower than the CPU. The GPU delegate failed to use OpenCL and used OpenGL as a fallback. The exception was MobileGoNogo which used OpenCL and performed better than on a single CPU core.

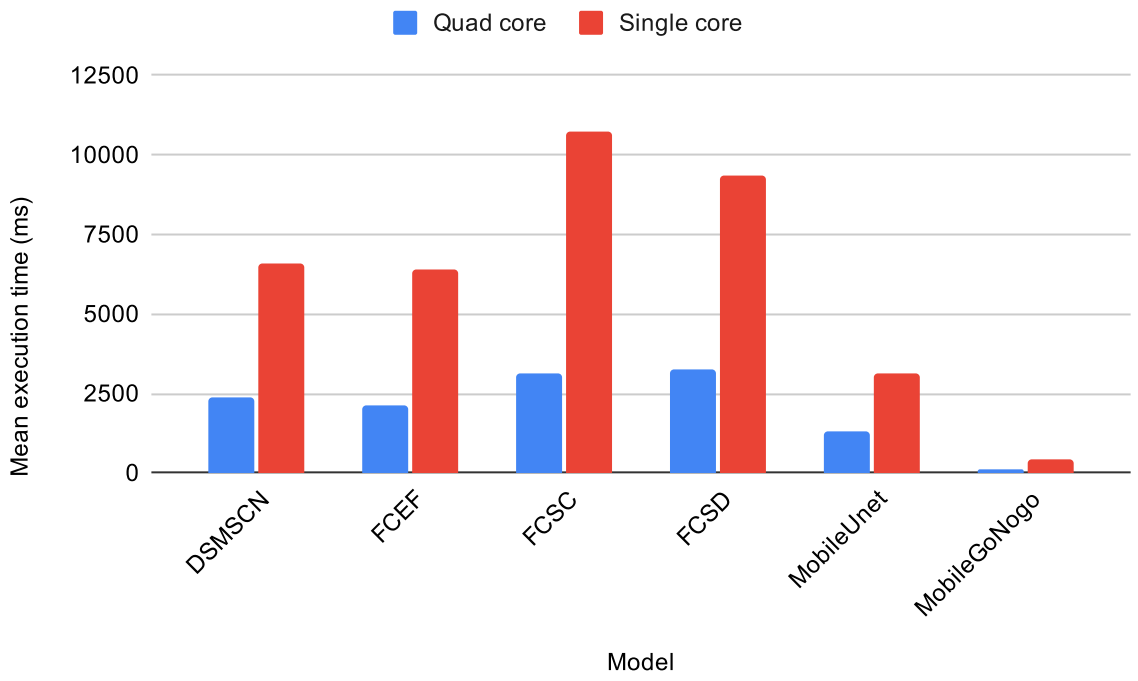


Figure 4.1.1: TFLite benchmarks

4.2 Change detection experiments

Change detection accuracy results are reported in Table 4.2.1 and Figure 4.2.1

Table 4.1.1: Runtime benchmarks of time (ms) and memory (MB) with XNNPACK

Model	CPU 1 core			CPU 4 core			GPU	
	time	SD	RAM	time	SD	RAM	time	SD
DSMSCN	6606	38	567	2405	49	566	2.54×10^5	228
FCEF	6369	6	376	2154	8	376	2.62×10^5	205
FCSC	10737	32	410	3147	303	410	3.63×10^5	n/a
FCSD	9345	21	262	3266	398	261	2.90×10^5	n/a
MobileUnet	3102	11	191	1313	132	191	9.50×10^4	n/a
MobileGoNogo	405	3	61	121	1	61	205	23

4.2.1 Direct classification

The OSCD dataset was used for reference in addition to SpaceNet 7. Models trained on the OSCD dataset showed similar accuracy as reported in [6], validating correct model architecture implementation.

Table 4.2.1: Direct classification model accuracy given by F1-score (F1), precision (Pr) and recall (Re)

Model	SpaceNet 7 (3 month)			SpaceNet 7 (12 month)			OSCD		
	F1	Pr	Re	F1	Pr	Re	F1	Pr	Re
DSMSCN	0.2696	0.3027	0.2429	0.4053	0.3950	0.4162	0.4342	0.4278	0.4408
FCEF	0.2581	0.2192	0.3136	0.4099	0.3716	0.4570	0.4364	0.5202	0.3758
FCSD	0.2544	0.2178	0.3057	0.4414	0.3905	0.5076	0.4805	0.3664	0.6978
FCSC	0.2730	0.2719	0.2742	0.4432	0.4168	0.4730	0.4597	0.4331	0.4899

4.2.2 Post-classification

Segmentation models xdxd and MobileUnet showed similar F1 accuracy scores. Precision and recall were fairly balanced in both models, with xdxd having slightly higher accuracy across metrics.

Change maps produced as the difference between segmentation masks were not nearly as accurate as the masks themselves. In both models, precision and F1 were close to 0. Recall however was almost as high for change maps as for underlying segmentation masks. These changes in accuracy indicate a problem of introducing many pseudo

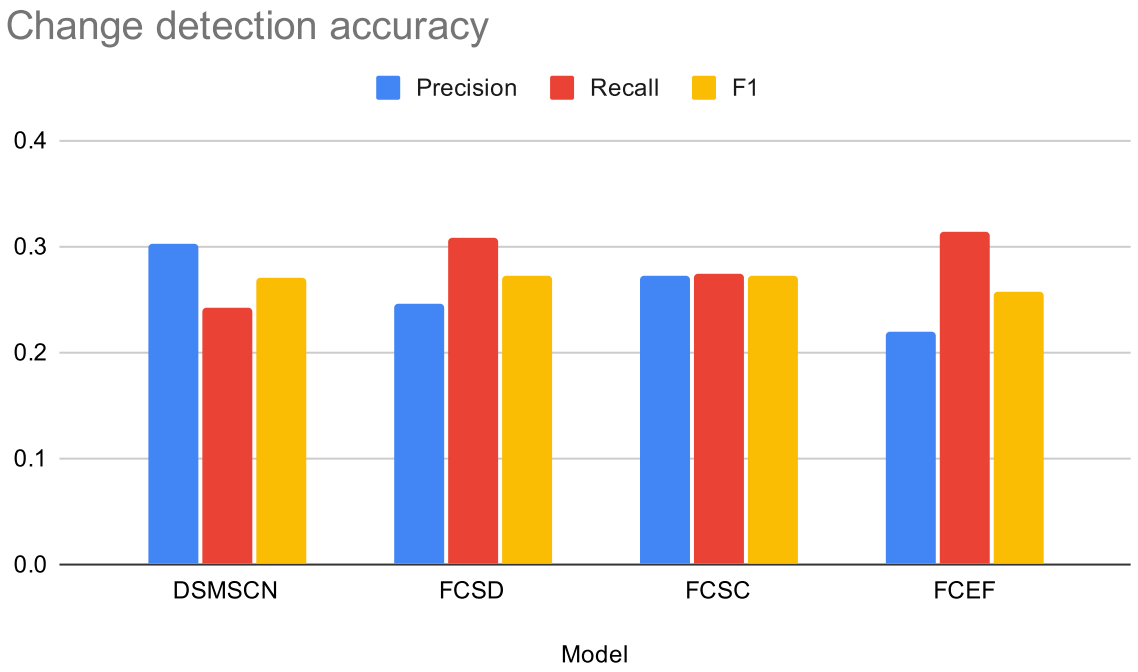


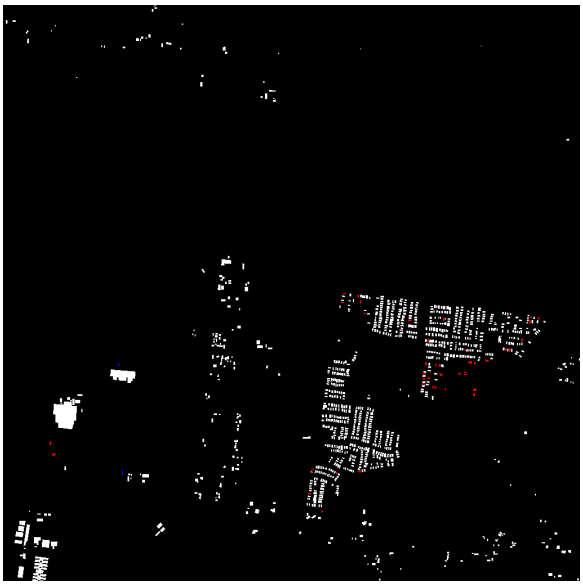
Figure 4.2.1: Model accuracy

changes, i.e. false positives.

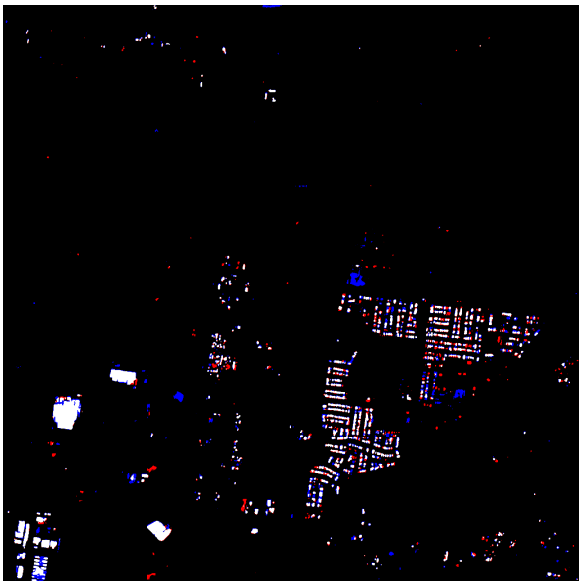
A major source of false positives was at the boundaries of objects. This effect can be seen in Figure 4.2.2c, most clearly in the lower left where the boundary of a larger building contains false changes. To the right where objects are small and dense, such boundary errors accumulated to a large number of false positives. This effect explains the loss in precision in post-classification for this data which contains many small objects. With recall being relatively high, the change maps capture many of the true changes. However, they are few and indistinguishable among the many false changes.

Table 4.2.2: Post classification model accuracy

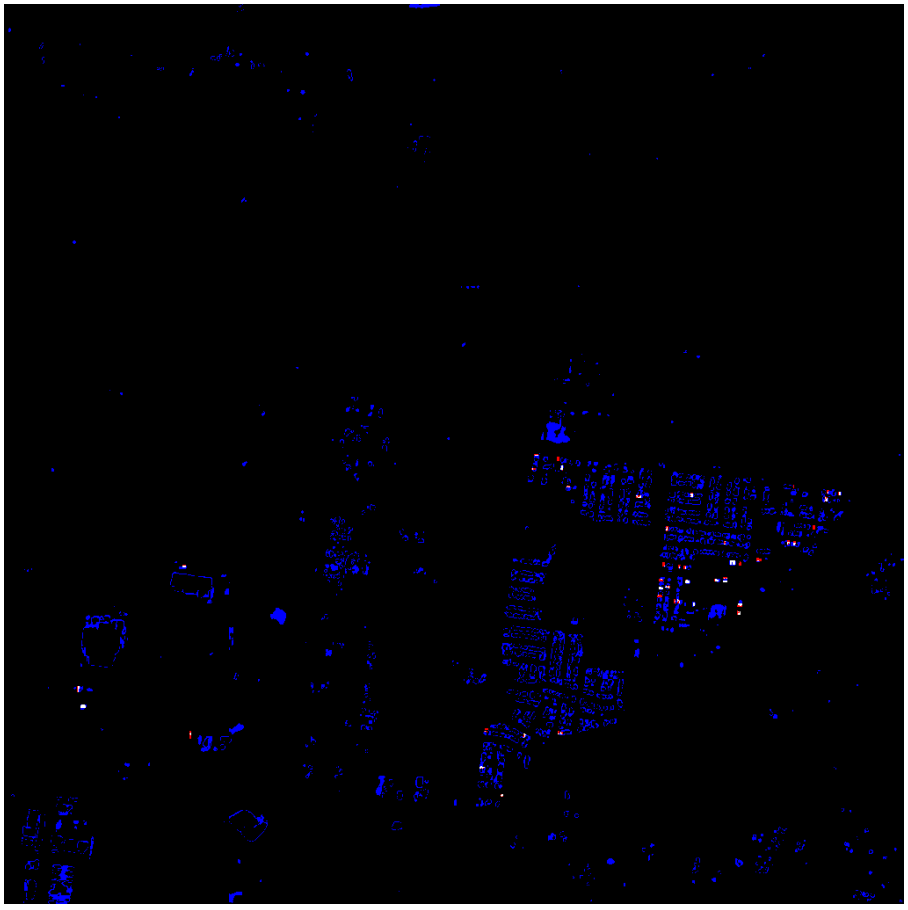
Model	Segmentation			Change map		
	F1	Precision	Recall	F1	Precision	Recall
xdxd	0.5829	0.6811	0.5217	0.0678	0.0470	0.4397
MobileUnet	0.4883	0.4730	0.5047	0.0367	0.0192	0.4299



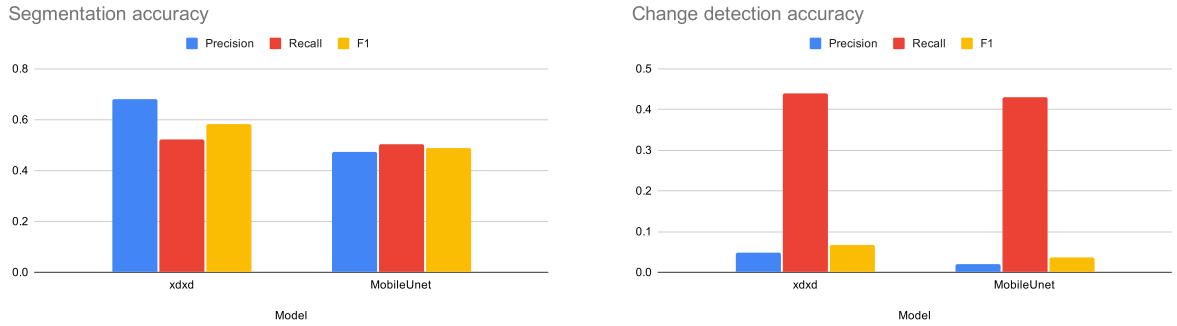
(a) True segmentation and changes.
Red: $0 \Rightarrow 1$, blue: $1 \Rightarrow 0$, white: no change



(b) Predicted segmentation and changes.
Red: $0 \Rightarrow 1$, blue: $1 \Rightarrow 0$, white: no change



(c) Predicted change map.
Red: false negative, blue: false positive, white: true positive



(a) Segmentation model accuracy

(b) post-classification accuracy

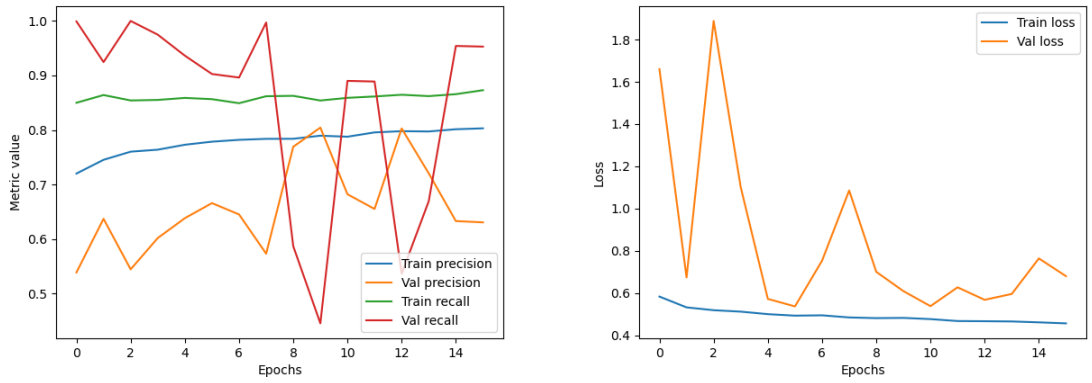
Figure 4.2.3: Accuracy of segmentation models when used for segmentation and change detection

4.2.3 Image patch classification

Accuracy results for patch classification are found in Table 4.2.3. Binary accuracy, Area Under Curve (AUC) and Precision-Recall Curve (PRC) are reported in addition to F1-score, precision, and recall reported for other models. Results are reported for 256×256 pixel patches without upscaling. The best model had a high recall and moderate precision at an F1 score of 0.7664.

Table 4.2.3: Image patch classification model accuracy

Model	F1	Precision	Recall	Accuracy	AUC	PRC
MobileGoNogo	0.7664	0.6659	0.9027	0.7228	0.8147	0.8032



(a) Metrics

(b) Loss

Figure 4.2.4: Training behaviour of MobileGoNogo

In Figure 4.2.4 the training behaviour of MobileGoNogo is found. Test loss in Figure 4.2.4b is unstable compared to training loss which decreases steadily although not

significantly. Likewise, the precision and recall for the training set increased fairly steadily while for test the metrics fluctuate in Figure 4.2.4a. When recall goes up, precision goes down and vice versa as expected from the trade-off between the metrics. Given the high variance between epochs, it is not safe to assume the model will predictably favor one or the other.

4.3 Model complexity

As seen in Table 4.3.1, MobileUnet was the smallest at almost half the size of the next smallest DSMSCN model. A relatively low parameter count was expected from a mobile optimized network architecture.

Table 4.3.1: Model parameter count as measure of complexity

Model	# params
MobileUnet	416,209
DSMSCN	768,129
FCEF	1,238,769
FCSD	1,248,017
FCSC	1,446,161
MobileGoNogo	3,535,617

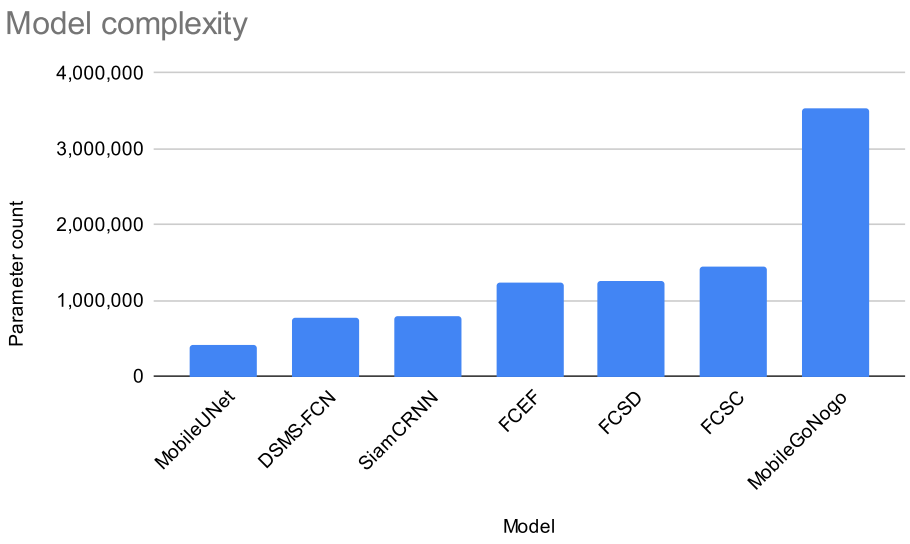


Figure 4.3.1: Model complexity as measured by number of parameters

All models are less than 14 MB in size as TFLite models. Model size storage is consequently negligible in relation to storage requirements for input and output data.

4.4 Storage analysis

The encoder output is smaller than the raw input in most models even without quantization, as seen in Table 4.4.1. However, all fully convolutional models compared use skip connections to provide spatial information to the decoder in order to accurately locate pixel-level changes. Due to skip connections, activation tensors at each convolutional layer of the encoder would have to be stored in addition to the encoder output. The total stored data is far greater than encoder output in that case. The smallest data to store is therefore the original image, i.e. $m_t = d_t$. Having to store the first image m_t of any pair until the next image m_{t+1} is available creates a storage buildup equal to the size of all images captured during the time frame of comparisons, which is at minimum the revisit period of the satellite.

Table 4.4.1: Encoder activation output size of 512 × 512 input with and without 8-bit integer quantization

Model	No quantization		8-bit quantization		Skip connections
	KB	Compression	KB	Compression	
DSMSCN	524	33%	131	83%	yes
FCEF	524	33%	131	83%	yes
FCSD	524	33%	131	83%	yes
FCSC	524	33%	131	83%	yes
MobileUnet	786	0%	197	75%	yes
MobileGoNogo	5.1	99.3%	1.3	99.8%	no

A segmentation map could to be stored instead of intermediate network data when using segmentation models in post-classification as done with models xdxd and MobileUnet. With segmentation maps being binary, a bitmask can be stored requiring only 1 bit per pixel. A full 1024 × 1024 bitmask would require $\frac{1024^2}{8} = 131072$ bytes, or ≈ 131Kb uncompressed. This size is substantially smaller than an original image as required in siamese networks. Such masks are highly compressible, requiring only 24 Kb as PNG files on average in the dataset.

There may be other methods of compression for segmentation maps, such as sparse matrices and vector data formats. The methods mentioned are not further analyzed in this work since the post-classification scheme did not show feasibility in terms of accuracy.

4.5 Use case scenarios

Assuming medium resolution like SpaceNet 7, each pixel covers ≈ 4m². A patch of size 512 × 512 consequently covers $512^2 \cdot 4 = 1048576$ m² or ≈ 1.05 km²

As deduced in section 4.4, direct classification models require storage of all images taken in a time frame. In the scenario with 10 days revisit time, the minimum time frame is 10 days.

The average size of 1024 × 1024 PNG ground images, PNG label masks and JPEG ground images in the dataset were calculated to be ≈ 1.7 MB, 0.024 MB and 0.34 MB, respectively.

Storage requirements in GB were calculated as

$$r = \frac{p \cdot s}{4 \cdot 1000}$$

where p is the processing capability (km²/day), s is the size of the image and r is the storage requirement per day. Division by 4 because each image covers ≈ 4 km² and 1000 to convert from MB to GB.

Based on inference times presented in Table 4.1.1, processing capabilities and storage requirements per day for the processed data is presented in Table 4.5.1. Notably, MobileGoNogo model can process an order of magnitude more image data than the other models but has next to lowest storage requirement nonetheless.

4.5.1 Scenario 1

A scenario with a single satellite with a 10 day revisit period would have storage requirements 10 times that of numbers reported in Table 4.5.1. The direct classification models DSMSCN etc. would require 123-187 GB of storage for a 10 day period with full size PNG or 23-36 GB for JPEG when processing at full speed constantly.

Table 4.5.1: Storage requirements per day of image processing for different image formats and models.

Model	km²/day	Storage requirement per day (Gb)	
		PNG[1]	JPEG
DSMSCN	3.77×10^4	16.73	3.21
FCEF	4.21×10^4	18.68	3.58
FCSC	2.88×10^4	12.79	2.45
FCSD	2.77×10^4	12.32	2.36
		Bitmask	PNG[2]
MobileUnet	6.90×10^4	2.26	0.41
		Feature vector	
MobileGoNogo	7.51×10^5	0.96	

Format sizes: PNG[1] 1.7 MB, JPEG 0.34 MB, Bitmask 0.131 MB, PNG[2] 0.024 MB, feature vector 0.0051 MB

4.5.2 Scenario 2

This scenario is a large constellation like PlanetScope with daily coverage of the planet at 200 million m² per day. Constellation size is assumed to be 150. An individual satellite in this scenario captures $\frac{2 \times 10^8}{150} = 1.33 \times 10^6$ km².

Table 4.5.2: Image processing capabilities at maximum resource utilization in relation to image capture capabilities

Model	Processing km² / day	Part processed
DSMSCN	37670	2.83%
FCEF	42059	3.15%
FCSC	28793	2.16%
FCSD	27738	2.08%
MobileUnet	68990	5.17%
MobileGoNogo	751480	56.36%

It is clear from Table 4.5.2 that processing speeds are a major bottleneck for satellites whose purpose is to monitor as much area as possible. Only patch classification like MobileGoNogo is approaching processing speeds required to handle the rate of image capturing data.

4.6 Summary

The various results are summarized in Table 4.6.1 to assess the technical feasibility of the whole system.

Table 4.6.1: Summary of judged feasibility metrics. See Table 4.6.2 for scale definitions.

Approach	Model	Accuracy	Speed	Storage	RAM	Resolution
Direct classification	DSMSCN	Poor	Poor	Poor	Poor	Very good
	FCSD	Poor	Poor	Poor	Moderate	Very good
	FCSC	Poor	Poor	Poor	Moderate	Very good
	FCEF	Poor	Poor	Poor	Moderate	Very good
Post classification	MobileUnet	Very poor	Poor	Good	Good	Very good
Patch classification	MobileGoNogo	Good	Good	Very good	Good	Very **poor**

Metric	Very poor	Poor	Moderate	Good	Very good
Accuracy (F1)	> 0.0	> 0.25	> 0.50	> 0.75	> 0.90
Speed (ms)	> 10000	> 1000	> 250	> 50	> 0
Storage (KB)	> 1000	> 500	> 100	> 10	> 0
RAM (MB)	> 1000	> 500	> 250	> 100	> 0
Resolution	$\geq 256 \times 256$	$\geq 64 \times 64$	$\geq 16 \times 16$	$\geq 8 \times 8$	$\geq 1 \times 1$

Table 4.6.2: Scale definitions for summary Table 4.6.1

Chapter 5

Conclusions

Results show that none of the models studied are technically feasible in all aspects for use in the proposed scenarios with current hardware. MobileGoNogo fulfills all technical criteria better than the other models except in spatial resolution where it differs significantly by being patch-based, as seen in Table 4.6.1. By drastically reducing the spatial resolution (patch vs pixel) it was able to overcome problems of the other methods. We conclude that the novel MobileGoNogo model is the most feasible choice for applications where spatial resolution is not critical.

Direct change detection of image pairs using siamese networks showed the best accuracy in terms of F1, precision, and recall at a pixel-level. None of the direct classification models allowed sparse representations to be stored in between capturing of images. The post-classification approach allowed such sparse representations but failed to fulfill the accuracy requirements of change maps due to a very high false positive rate.

No model was fast enough during inference to process collected image data in the proposed scenarios on the ix2000 compute modules. MobileGoNogo was about half as fast as required for full coverage and an order of magnitude faster than the other models. Memory usage was from about 9% to 27% of capacity. Usage is substantial but not a limiting factor. Model sizes were less than 14 Mb which makes it negligible in affecting storage requirements.

Hardware architecture and tooling matters greatly for inference time on edge devices. With on-board computers for satellites having diverse architectures, this is concluded to be a key factor besides raw compute power in meeting inference constraints.

Quantization and GPU inference, in general, are proven techniques for fast and efficient ML but failed to perform on this setup because of architecture-tool mismatch. With a diverse set of software stacks and hardware configurations amongst OBCs it is important to consider software availability in determining technical feasibility of the system. Using an FPGA or a VPU device like Movidius and selecting for compatible models could be ways to overcome the performance shortcomings of the stack investigated.

5.1 Discussion

The dataset is challenging when changes are few and small. It is especially difficult for post-classification methods that have boundary errors. With a large portion of the pixels being boundaries, such errors accumulate to a noisy picture.

GPU results were unexpected. It is well known that, in general, GPUs accelerate CNN models. However, small models may be faster on CPUs. The TFLite GPU delegate does not work well on this particular hardware. TFLite is designed for ARM devices but the development machine uses x86 architecture. Using 16 bit floating point operations is not supported on the x86 GPU as compared to ARM. Therefore 16-bit quantization had no positive effect on inference times. We hypothesize that when utilizing the GPU, integer dequantization was performed on the CPU such that it switched back and forth between CPU and GPU causing it to not perform well. Knowing that this particular GPU was not suitable for most of the models does not conclude that GPUs are a poor choice for CD models, especially since the compatible MobileGoNogo model did show a significant increase in performance on the GPU.

Pixel-based models were too slow despite the quad core CPU which is considered to be high performing in OBCs. As a result, the models can be considered too slow for most OBCs. Even if the models would be fast enough for processing, energy consumption and cooling could be bottlenecks. Running in 4 thread mode is not the most energy efficient way but perhaps necessary for performance reasons.

One explanation for an increased variance on quad core configuration can be higher sensitivity to other concurrent processes on the hardware when requesting maximum capacity from all cores. Some models may parallelize better than others, since the drastic increase in variance of compute time on quad core was only observed for some

models but stable for others.

5.1.1 Future Work

As seen in results, post-classification methods suffered from boundary errors due to a high number of small objects. We believe detecting changes in larger objects such as farmlands to work significantly better when using post-classification. We further find it of interest to apply all methods used in this work to other types of change detection than buildings.

One could use a Very High Resolution (VHR) dataset such as SkySat 0.5 m resolution. We hypothesize it to be significantly easier for models to accurately find buildings and their changes with higher resolution.

We believe it would be useful to extend the feasibility evaluation with a detailed energy consumption analysis. Like compute power, energy availability can vary significantly from mission to mission. Energy consumption is naturally linked directly to inference time studied in this work. However, the magnitudes of energy consumption and availability need to be studied to determine feasibility with energy in mind.

A sparse difference image could be produced In the case of frequent imagery in a sun-synchronous orbit where image pair are likely to be similar on a pixel level. This image could then be used as input to a model which could perhaps determine whether this difference image has a change of interest or not. Hopefully, by having less pixels as input the model could be faster than models getting fed a full image pair.

Post processing

The SpaceNet 7 challenge winner [28] concluded that most of the innovation and reason for its success was its post processing methods. Assumptions were made for behavior of buildings specifically and thus it does not apply to change detection generally. It is important for this problem nonetheless. The methods proposed are known to be successful at change detection in longer time series. Evaluation of inference time and storage requirements remains to be performed.

As seen in results, change maps resulting from post-classification are noisy with a high degree of false positives and low precision. The method of post-classification is still interesting for future work due to its low storage requirements, high spatial resolution,

and high accuracy in segmentation models. Post processing or more complex algorithms for creating change maps from segmentation masks could potentially mitigate the problem of false positives and thus increase accuracy. We identify the following method candidates for future work:

- Create polygons from segmentation and look for overlap. Use overlap threshold to determine change or not.
- Probabilistic model using assumptions (priors) such as existing buildings being unlikely to change or disappear
- Remove pixels without neighbors, or other means of salt and pepper noise reduction.
- Compare series of more than two images and only select consistent changes in the series.

MobileGoNogo

The MobileGoNogo model or other patch classification models can preferably be tuned for higher recall and lower precision in the precision-recall trade-off. It can be argued that false positives are not so detrimental as false negatives in the use case. Downlinking images that have no changes is perhaps unnecessary but *not* downlinking images that *have* changes jeopardizes the functionality of the system. In effect, having a false alarm can arguably be considered better than missing a true alarm.

A plausible reason for sub-optimal accuracy in the MobileGoNogo model is that virtually all spatial information is lost at the final global averaging pooling layer. Rahman et al. [16] preserve spatial information by using entire CNN blocks as input for decision network whereas MobileGoNogo uses the pooled feature vector as commonly used in image classification. As stated previously, the reason for the pooling architecture in MobileGoNogo is drastically reduced storage requirements. Connecting the entire last CNN block to the decision network is possible but impractical. It could however test the idea of accuracy loss due to lost spatial information.

We hypothesize that patch pairs containing a change from no buildings to one or more buildings are most accurately classified. When the first image contains a number of buildings and the second image contains a larger number of buildings, however, it may

be hard for the network to pick up on the change since both images would correctly be classified as having buildings. This hypothesis is not experimentally verified.

The model may be overfitting due to a relatively low number of samples in the dataset. SpaceNet 7 has 2389 images as compared to 11700 used in [16]. For classification networks like MobileNetV2, which MobileGoNogo is based on, ImageNet dataset is often used as a standard containing over 14×10^6 images. Although the number of images were the same for patch and pixel-based models, the patch-based approach may suffer from a small number of labeled samples. A pixel-level model effectively has $256^2 = 65536$ labels per 256×256 pixel patch for which a patch-based approach has only one label. We suggest training patch-based models on a far larger dataset to avoid potential overfitting. Such a dataset could potentially be generated as in [16].

A patch-based approach does not necessarily lose all spatial information. The smaller the patch size, the more spatial information is preserved with the trade-off of more processing time. For example, splitting and image into 16 patches and finding a change in just one patch would yield a bandwidth reduction of factor 16. We propose a systematic study of varying patch sizes to find the relation between patch size and all other metrics studied in this work. Knowing such relationship would assist in finding an optimal patch size given the constraints of a particular use case scenario.

Another option for efficiently and dynamically finding a suitable patch size could be a divide-and-conquer approach. A patch (initially the whole image) is split into 4 sub patches recursively for further CD as long as the patch is labeled as changed. Each recursion reduces the patch size by a factor of 4. The final patch size would be the smallest size the network is able to detect a change in, assuming accuracy is preserved. For images with few or small changes, this could lead to small patch sizes which in turn lead to higher spatial resolution and bandwidth savings. MobileGoNogo would have to be trained for each patch size individually due to the fully connected layers in the decision network requiring a fixed size. With the model's high parameter count in relation to the other models, training and storing multiple versions could become a practical problem.

5.1.2 Final Words

On-board change detection using ML is a challenging problem. Intelligent on-board image processing is at an early stage but shows promise and change detection can be

one kind of such processing. However, more research and development is needed in OBCs, ML-based change detection and the intersection of the two, before on-board change detection becomes common practice in the industry.

Bibliography

[1] Badrinarayanan, Vijay, Kendall, Alex, and Cipolla, Roberto. “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39.12 (Dec. 1, 2017), pp. 2481–2495. ISSN: 0162-8828, 2160-9292, 1939-3539. DOI: 10.1109/TPAMI.2016.2644615. URL: https://ieeexplore.ieee.org/document/7803544/ (visited on 03/19/2021).

[2] Bruhn, Fredrik C., Tsog, Nandinbaatar, Kunkel, Fabian, Flordal, Oskar, and Troxel, Ian. “Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space”. In: *CEAS Space Journal* 12.4 (Dec. 2020), pp. 551–564. ISSN: 1868-2510. DOI: 10.1007/s12567-020-00321-9. URL: https://doi.org/10.1007/s12567-020-00321-9.

[3] Chen, H., Wu, C., Du, B., Zhang, L., and Wang, L. “Change Detection in Multisource VHR Images via Deep Siamese Convolutional Multiple-Layers Recurrent Neural Network”. In: *IEEE Transactions on Geoscience and Remote Sensing* 58.4 (2020), pp. 2848–2864. DOI: 10.1109/TGRS.2019.2956756.

[4] Chen, Hongruixuan, Wu, Chen, Du, Bo, and Zhang, Liangpei. “Change Detection in Multi-temporal VHR Images Based on Deep Siamese Multi-scale Convolutional Networks”. In: *arXiv:1906.11479 [cs, eess]* (July 10, 2020). arXiv: 1906.11479. URL: http://arxiv.org/abs/1906.11479 (visited on 03/04/2021).

[5] Chen, Hongruixuan, Wu, Chen, Du, Bo, and Zhang, Liangpei. “DSDANet: Deep Siamese Domain Adaptation Convolutional Neural Network for Cross-domain Change Detection”. In: *arXiv:2006.09225 [cs]* (June 16, 2020). arXiv: 2006.09225. URL: http://arxiv.org/abs/2006.09225 (visited on 07/26/2021).

[6] Daudt, Rodrigo Caye, Saux, Bertrand Le, and Boulch, Alexandre. “Fully Convolutional Siamese Networks for Change Detection”. In: *arXiv:1810.08462 [cs]* (Oct. 19, 2018). arXiv: 1810.08462. URL: http://arxiv.org/abs/1810.08462 (visited on 02/28/2021).

[7] EarthData, NASA. *What is Remote Sensing?* https://earthdata.nasa.gov/learn/backgrounders/remote-sensing.

[8] Etten, Adam Van, Hogan, Daniel, Martinez-Manso, Jesus, Shermeyer, Jacob, Weir, Nicholas, and Lewis, Ryan. *The Multi-Temporal Urban Development SpaceNet Dataset*. 2021. arXiv: 2102.04420 [cs.CV].

[9] George, A. D. and Wilson, C. M. “Onboard Processing With Hybrid and Reconfigurable Computing on Small Satellites”. In: *Proceedings of the IEEE* 106.3 (2018), pp. 458–470. DOI: 10.1109/JPROC.2018.2802438.

[10] Khelifi, L. and Mignotte, M. “Deep Learning for Change Detection in Remote Sensing Images: Comprehensive Review and Meta-Analysis”. In: *IEEE Access* 8 (2020), pp. 126385–126400. DOI: 10.1109/ACCESS.2020.3008036.

[11] Kingma, Diederik P. and Ba, Jimmy. “Adam: A Method for Stochastic Optimization”. In: *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*. Ed. by Yoshua Bengio and Yann LeCun. 2015. URL: http://arxiv.org/abs/1412.6980.

[12] Loop, Humans In The. *Semantic segmentation dataset*. Accessed on 2021-07-24. 2021. URL: https://humansintheloop.org/resources/datasets/semantic-segmentation-dataset/.

[13] *NeurIPS Competition Track*. 2020. URL: https://neurips.cc/Conferences/2020/CompetitionTrack (visited on 08/13/2021).

[14] Peng, Daifeng, Zhang, Yongjun, and Guan, Haiyan. “End-to-End Change Detection for High Resolution Satellite Images Using Improved UNet++”. In: *Remote Sensing* 11.11 (2019). ISSN: 2072-4292. DOI: 10.3390/rs11111382. URL: https://www.mdpi.com/2072-4292/11/11/1382.

[15] Pierre-Philippe Mathieu, Sveinung Loekken. *Towards a European AI4EO Research and Innovation Agenda. ESA Φ-lab workshop proceedings*. Issue 1, Rev 3. Sept. 2018.

[16] Rahman, Faiz, Vasu, Bhavan, Cor, Jared Van, Kerekes, John, and Savakis, Andreas. “SIAMESE NETWORK WITH MULTI-LEVEL FEATURES FOR PATCH-BASED CHANGE DETECTION IN SATELLITE IMAGERY”. In: *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. Anaheim, CA, USA: IEEE, Nov. 2018, pp. 958–962. ISBN: 978-1-72811-295-4. DOI: 10.1109/GlobalSIP.2018.8646512. URL: https://ieeexplore.ieee.org/document/8646512/ (visited on 03/04/2021).

[17] Ronneberger, Olaf, Fischer, Philipp, and Brox, Thomas. “U-Net: Convolutional Networks for Biomedical Image Segmentation”. In: *Medical Image Computing and Computer-Assisted Intervention – MICCAI 2015*. Ed. by Nassir Navab, Joachim Hornegger, William M. Wells, and Alejandro F. Frangi. Cham: Springer International Publishing, 2015, pp. 234–241. ISBN: 978-3-319-24574-4.

[18] Sandler, Mark, Howard, Andrew, Zhu, Menglong, Zhmoginov, Andrey, and Chen, Liang-Chieh. “MobileNetV2: Inverted Residuals and Linear Bottlenecks”. In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition. ISSN: 2575-7075. June 2018, pp. 4510–4520. DOI: 10.1109/CVPR.2018.00474.

[19] *Satellite Imagery and Archive | Planet*. URL: https://www.planet.com/products/planet-imagery/ (visited on 03/19/2021).

[20] *Sentinel-2 - Missions - Sentinel Online - Sentinel*. URL: https://sentinel.esa.int/web/sentinel/missions/sentinel-2 (visited on 03/05/2021).

[21] Shelhamer, Evan, Long, Jonathan, and Darrell, Trevor. “Fully Convolutional Networks for Semantic Segmentation”. In: *arXiv:1605.06211 [cs]* (May 20, 2016). arXiv: 1605.06211. URL: http://arxiv.org/abs/1605.06211 (visited on 03/11/2021).

[22] Shi, Wenzhong, Zhang, Min, Zhang, Rui, Chen, Shanxiong, and Zhan, Zhao. “Change Detection Based on Artificial Intelligence: State-of-the-Art and Challenges”. In: *Remote Sensing* 12.10 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12101688. URL: https://www.mdpi.com/2072-4292/12/10/1688.

[23] *SpaceCloudTM iX5100 Solution – Radiation Tolerant Intelligent Data Processing in Space*. URL: https://unibap.com/wp-content/uploads/2020/11/spacecloud-ix5100-datasheet.pdf (visited on 03/19/2021).

[24] Sudre, Carole H., Li, Wenqi, Vercauteren, Tom, Ourselin, Sebastien, and Jorge Cardoso, M. “Generalised Dice Overlap as a Deep Learning Loss Function for Highly Unbalanced Segmentations”. In: *Lecture Notes in Computer Science* (2017), pp. 240–248. ISSN: 1611-3349. DOI: 10.1007/978-3-319-67558-9_28. URL: http://dx.doi.org/10.1007/978-3-319-67558-9_28.

[25] *TFLite Model Benchmark Tool with C++ Binary*. URL: https://github.com/tensorflow/tensorflow/tree/master/tensorflow/lite/tools/benchmark (visited on 07/13/2021).

[26] Tsog, N., Behnam, M., Sjödin, M., and Bruhn, F. “Intelligent data processing using in-orbit advanced algorithms on heterogeneous system architecture”. In: *2018 IEEE Aerospace Conference*. 2018, pp. 1–8. DOI: 10.1109/AERO.2018.8396536.

[27] *Unibap — SpaceCloud® Framework - manual*. URL: https://unibap.com/en/support/spacecloud-framework-manual/ (visited on 03/19/2021).

[28] Van Etten, Adam and Hogan, Daniel. “The SpaceNet Multi-Temporal Urban Development Challenge”. In: *arXiv:2102.11958 [cs]* (Feb. 23, 2021). arXiv: 2102.11958. URL: http://arxiv.org/abs/2102.11958 (visited on 03/15/2021).

[29] Yang, Kunping, Xia, Gui-Song, Liu, Zicheng, Du, Bo, Yang, Wen, Pelillo, Marcello, and Zhang, Liangpei. “Asymmetric Siamese Networks for Semantic Change Detection in Aerial Images”. In: *IEEE Transactions on Geoscience and Remote Sensing* (2021), pp. 1–18. DOI: 10.1109/TGRS.2021.3113912.

[30] Yang, S., Liu, Z., Gao, Q., Gao, Y., and Feng, Z. “Extreme Self-Paced Learning Machine for On-Orbit SAR Images Change Detection”. In: *IEEE Access* 7 (2019), pp. 116413–116423. DOI: 10.1109/ACCESS.2019.2934983.

[31] You, Yanan, Cao, Jingyi, and Zhou, Wenli. “A Survey of Change Detection Methods Based on Remote Sensing Images for Multi-Source and Multi-Objective Scenarios”. In: *Remote Sensing* 12.15 (2020). ISSN: 2072-4292. DOI: 10.3390/rs12152460. URL: https://www.mdpi.com/2072-4292/12/15/2460.

TRITA-EECS-EX-2021:750