Low-power neural networks for semantic segmentation of satellite images

Gaétan Bahl[1,2], Lionel Daniel[2], Matthieu Moretti[2], Florent Lafarge[1]
[1]Université Côte d'Azur - INRIA, [2]IRT Saint-Exupéry
{gaetan.bahl, florent.lafarge}@inria.fr, {lionel.daniel, matthieu.moretti}@irt-saintexupery.com

Abstract

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

1. Introduction

The semantic segmentation of images, which consists in assigning a semantic label to each pixel of an image, is a long standing problem in Computer Vision. The popularization of Convolutional Neural Networks (CNN) has led to a lot of progress in this field. Fully Convolutional Networks (FCN) [23] and related architectures, such as the popular U-Net [31], outperformed traditional methods in terms of accuracy while being easy to use.

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

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

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

Some efficient architectures, such as ESPNet [24], and convolution modules [12] have been proposed for phones and autonomous vehicles. These architectures are however too complex to be exploited on devices with lower power budget and computational resources. For instance, this is the case of FPGA cards embedded into system-on-chips (SoCs) on satellites, which have limited floating-point capabilities and are restricted by their amount of hardware logic.

In this paper, we present two highly-compact neural network architectures for semantic segmentation, which are up to 100 000 times less complex than state-of-the-art architectures while approaching their accuracy. We propose two fully-convolutional neural networks, C-FCN and C-UNet, which are compact versions of the popular FCN [23] and U-Net [31] architectures. To decrease the complexity of existing networks, our main ideas consist in exploiting lightweight encoders and decoders with depth-wise separable convolutions and decreasing memory usage with the removal of skip connections between encoder and decoder. Our architectures are designed to be implemented on a basic FPGA such as the one featured on the Intel Altera Cyclone V family of SoCs. We demonstrate the potential of our solutions in the case of binary segmentation of remote sensing images, in particular for extracting clouds and trees from RGB satellite images. Our methods reach 95% accuracy in cloud segmentation on 38-Cloud [25] and CloudPeru2

[28] datasets, and 83% in forest segmentation on EOLearn Slovenia 2017 dataset.

2. Related works

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

2.1. Neural networks for semantic segmentation

Many deep learning methods have been proposed to improve performance of segmentation networks, in terms of accuracy on popular benchmarks such as [6, 21, 29] and speed, with the goal of real-time semantic segmentation [32]. These methods are compared in recent surveys [9, 7, 32] without studying the networks from a complexity point of view.

Encoder-decoders. Most of the popular segmentation neural networks, such as U-Net [31], SegNet [2], or DeepLabv3+ [3] adopt an encoder-decoder architecture. The encoder part of the network, also called backbone, is typically a Deep Convolutional Neural Network composed of multiple stages of convolution operations, separated by pooling operations. This allows the encoder to capture high-level features at different scales. It is common to use an existing CNN architecture such as ResNet-101 [11], VGG16 [34], or MobileNet [12] as a backbone, since pre-trained weights on databases such as ImageNet [15] are available for transfer learning. However, while achieving great accuracy, these encoders are complex and not suited for inference on edge devices. Decoders are in charge of upsampling the result to get an output that has the same size as the original image. Deconvolutions, also called transposed convolutions or up-convolutions, introduced by H. Noh *et al*. [30] in the context of semantic segmentation, are used to learn how images should be upscaled.

Skip connections. Skip connections are often used between convolution blocks, i.e. short skips, in architectures such as residual networks [11], and/or between the encoder and the decoder, i.e. long skips, in architectures such as U-Net [31]. These connections are performed by concatenating feature maps from different parts of the network in order to retain some information. In the case of segmentation networks, skip connections are used to keep high-frequency information (e.g. corners of buildings, borders of objects) to obtain a more precise upscaling by the decoder. While skip connections can yield good results in practice, they are often memory consuming, especially the long skips, as they require feature maps to be kept in memory for a long time. Consequently, they cannot easily be used on edge devices with limited memory. In particular, we show in Section 4 that they are not useful for our use cases.

2.2. Neural networks on edge devices

Efficient convolution modules. Several low-complexity convolution modules have been developed, with the goal of reducing power consumption and increase inference speed on low-end or embedded devices, such as phones. This is the case of Inception [36], Xception [4], ShuffleNet [39], or ESP [24]. These modules are based on the principle of convolution factorization and turn classical convolution operations into multiple simpler convolutions. S. Mehta *et al*. provide a detailed analysis of these modules in [24]. However, we cannot use them as they still feature too many trainable parameters and often require to store the results of multiple convolution operations performed in parallel or short skip connections. In contrast, we use Depth-wise Separable Convolutions, introduced in [33] and used in MobileNets [12].

Neural Network simplification Several techniques have been developed to simplify neural networks for systems with tight resource constraints. Training and inference using fixed-point or integer arithmetic and quantization is an active research topic [20, 14, 37], since floating-point operations are costly on many embedded systems such as FPGAs, and trade-offs have to be made between accuracy and speed. Neural network pruning, on the other hand, is the process of removing some weights [18] or entire convolution filters [19, 27] and their associated feature maps in a neural network, in order to extract a functional "sub-network" that has a lower computational complexity and similar accuracy. A lot of work has been done on enabling and accelerating NN inference on FPGA [1], with tools being released to automatically generate HDL code for any neural network architecture [10], with automated use of quantization and other simplification techniques. These approaches are orthogonal to our work as they could be applied to any neural network.

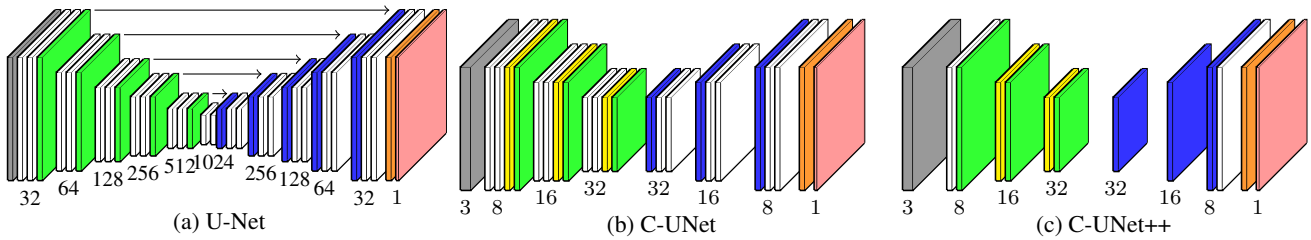
3. Proposed architectures

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

3.1. C-UNet

As illustrated in Figure 1, C-UNet is a compact version of the popular U-Net architecture [31].

To create a shallower network, three convolution stages have been removed with respect to the original architecture. Removing stages strongly reduces the computational cost as the number of feature maps is typically multiplied by a factor 2 at each stage. For instance, U-Net [31] has 1024 feature maps at the last encoder stage. A standard 3x3 convolution at this stage uses $3\times3\times1024\times1024+1024 = 9438208$



(a) U-Net (b) C-UNet (c) C-UNet++

Figure 1: U-Net [31] compared to C-UNet and C-UNet++. Gray: input, white: conv3x3 + ReLU, yellow: depthwise separable conv3x3 + ReLU, green: 2x2 max pool, orange: conv1x1 sigmoid, blue: 2x2 deconvolution, red: output, arrows: skip connections. Numbers indicate the number of feature maps at each stage.

parameters. The number of stages to remove has been chosen after experimental validation, 3 being a good compromise between field of view and number of parameters. Indeed, we found experimentally that we could train a classifier to detect objects such as clouds and trees in 28x28 pixel images with good accuracy. The encoder in C-UNet has a receptive field of 50x50 pixels and is consequently more than capable of reaching the same accuracy.

Some of the standard convolution layers have also been replaced by depth-wise separable convolutions. Introduced in [12], they allow a classical convolution to be split into a depth-wise convolution and a point-wise convolution, to use the fact that inter-channel features are often not related to spatial features. Since the convolution kernel is the same for all input feature maps in a depth-wise convolution, this allows us to reduce the number of learned parameters significantly. For example, a standard 3x3 convolution with 16 input and output feature maps has 2320 trainable weights, while a depth-wise separable convolution of the same size only has 416 trainable weights.

The number of filters per convolution has been chosen so that C-UNet has 51 113 parameters, ie around 500 times less than the architecture implemented in [25]. C-UNet then has 8 filters per convolution at the first stage and this number is doubled at every stage in the encoder, and divided by two at every stage in the decoder, as in the original U-Net architecture [31].

We also propose a variant of C-UNet, called C-UNet++, with an even more compact architecture. C-UNet++ contains 9 129 parameters, ie 3000 times less parameters than the architecture implemented in [25]. As illustrated in Figure 1, the number of convolution layers has been reduced to one per stage, and only the first stage has standard convolutions. This gives C-UNet++ a receptive field of 22x22 pixels.

Note that skip connections between corresponding stages in encoder and decoder parts could be considered. However, we show in Section 4 that they are not necessarily useful, as adding them may not significantly increase performance and may require a lot of memory depending on the input image size.

3.2. C-FCN

The second proposed architecture, called C-FCN, relies upon a small CNN and a bilinear upsampler, as illustrated in Figure 2. The encoder has 3 stages, with a single convolution per stage. A 1x1 convolution then generates a class heatmap which is then upsampled by the bilinear upsampler. C-FCN uses depthwise separable convolutions to maximize the number of convolution filters we can use within our parameter budget, which is of 1 438 parameters for this architecture.

We also propose a variant of C-FCN with the same depth, called C-FCN++, which is designed to be the smallest viable architecture with only 273 parameters. The first layer is an atrous convolution layer with a dilation rate of 2, which broadens the receptive field of the network a little bit [38].

The number of filters of C-FCN and its variant C-FCN++ are showed in Table 1.

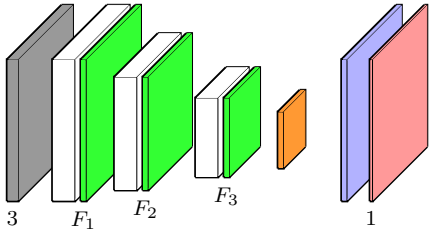


Figure 2: C-FCN and C-FCN++ architectures. Gray: input, white: conv3x3 ReLU, green: 2x2 max pool, orange: conv1x1 sigmoid, blue: 4x4 bilinear upscaling.

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

Table 1: Characteristics of C-FCN and C-FCN++. DW = usage of depthwise convolutions. F_x is the number of convolution filters and feature maps at stage x.

4. Experiments on cloud segmentation

In this section, we evaluate the performance of our architectures for cloud extraction in RGB remote sensing images. Performing real-time cloud segmentation directly onboard and removing useless cloud-covered images can allow significant bandwidth, storage and computation time savings on the ground. Accurate detection of clouds is not an easy task, especially when a limited number of spectral bands

is available, as clouds share the same radiometric properties as snow, for example. Traditional cloud segmentation methods are either threshold-based or handcrafted, with the most popular ones being Fmask [43] and Haze Optimized Transform [40].

M. Hughes *et al*. [13] were, to our knowledge, the first to use a neural network-based approach for cloud segmentation in Landsat-8 images. However, convolutional neural networks were not used. A lot of recent work has been done on cloud segmentation using CNN [25, 26, 28, 5, 22]. However, all these methods use computationnaly-heavy architectures on hyperspectral images and are designed to be used with powerful hardware on the ground, and thus, are not compatible with our use-case. Cloud segmentation using neural networks has already been integrated in an ARTSU CubeSat mission by Z. Zhang *et al*. [41]. S. Ghassemi *et al*. [8] have proposed a small strided U-Net architecture for onboard cloud segmentation. Their architectures are still too big for our use cases and are designed to work on 4-band images (RGB + NIR). Nevertheless, we include a 3-band implementation of MobUNet [41], MobDeconvNet [41] and "Plain+" strided U-Net [8] architectures in our comparison.

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

We use the Cloud-38 dataset, introduced by S. Mohajerani *et al*. [26]. The dataset is composed of 4-band Landsat-8 images with a 30m resolution. We only use RGB bands. Since some of the training scenes have black borders in the original dataset, we remove all the patches that have more than 50% of black pixels. The training and validation set is composed of 4 821 patches of size 384x384. The testing set is composed of 9 200 patches.

We also use the CloudPeru2 dataset, introduced by G. Morales *et al*. [28]. This dataset is composed of 4-band PERUSAT-1 images with a 3m resolution. It features 22 400 images of size 512x512. We use 10% of these images for testing. Test images have been chosen randomly and are the same for all of our tests.

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

Models are trained using a round-based scheme. Training sets are randomly shuffled and split into training (85%), and validation (15%) sets at the beginning of each round. Within a round, all models are then trained on the same training and validation sets for a maximum of 150 epochs. At the end of all rounds, the best network for each architecture is selected using the testing set.

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

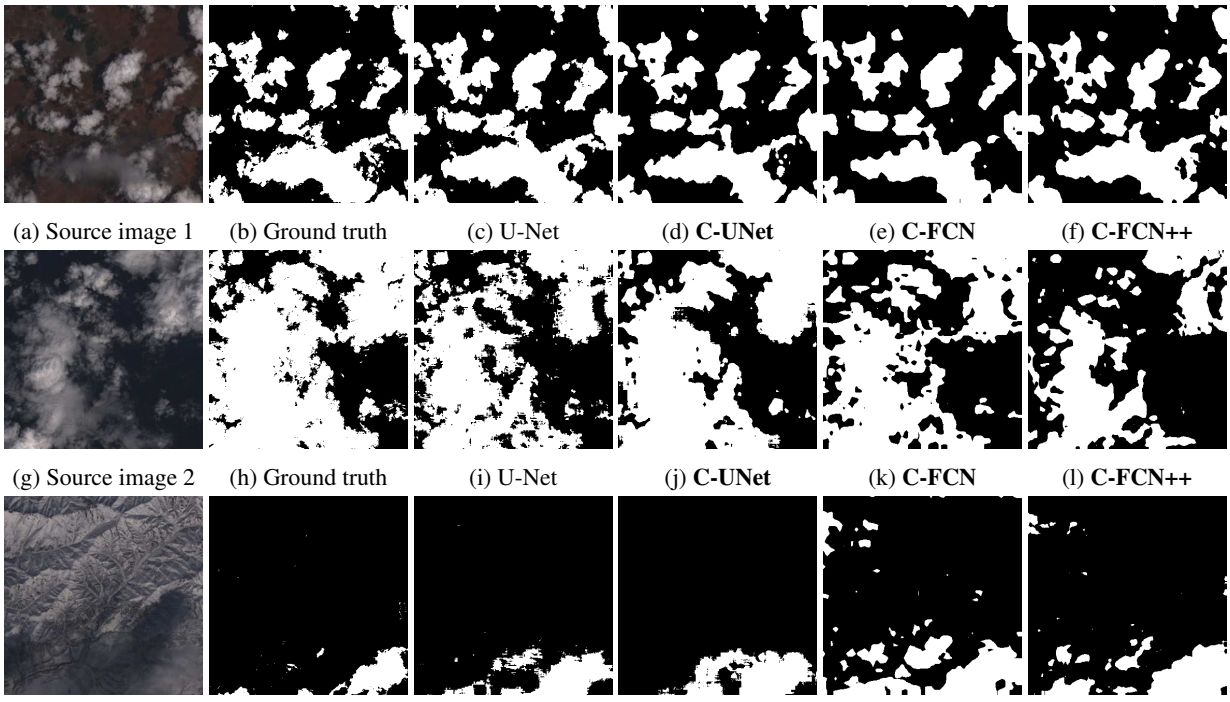
This training scheme takes around 10h per architecture, per round, per dataset on an Nvidia GTX 1080 Ti graphics card, which brings our total training time to around 1160h. Training could have been faster, as the card's VRAM was barely being used when training really small models, but we wanted to use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

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

- U-Net for cloud segmentation, as presented by S. Mohajerani *et al*. [25], which we use as a "baseline" for comparison, since U-Net, originally presented by Ronneberger *et al*. [31] for the segmentation of medical images is a well-known and easy to implement architecture that has proved useful for many segmentation tasks,
- ESPNet_A, introduced by S. Mehta *et al*. [24], which is a state-of-the-art segmentation network, in terms of efficiency and accuracy. We use $K = 5, \alpha_2 = 2, \alpha_3 = 5$.
- MobUNet and MobDeconvNet, introduced by Z. Zhang *et al*. [41], are small versions of U-Net [31] and Deconv-Net [30] for onboard cloud segmentation, that use depth-wise separable convolutions [12],
- StridedUNet, introduced by Ghassemi *et al*. [8] for onboard cloud segmentation, which uses strided convolutions for downsampling,
- LeNetFCN, an FCN variation of the well-known LeNet-5 architecture [17], which was originally created for written number classification. We replace the final Fully-Connected (Dense) layers by a 1x1 2D Convolution with a sigmoid activation, in order to output a segmentation map. LeNetFCN architecture looks like the "C-FCN" architecture (see Fig.2) with $F_1 = 3, F_2 = 5$, except it uses 5x5 convolutions instead of 3x3 convolutions.

We evaluate models using standard segmentation metrics: Total Accuracy, Precision, Recall, Specificity and Jaccard Index (IoU), as defined in [26].

Qualitative results. Fig. 3 shows inference results of our networks on three image patches taken from the 38-Cloud test set, compared to the ground truth and U-Net. Our networks provide a visually good result, even on difficult terrains such as snow. The segmentation masks produced appear smooth, especially for C-FCN/C-FCN++ because of

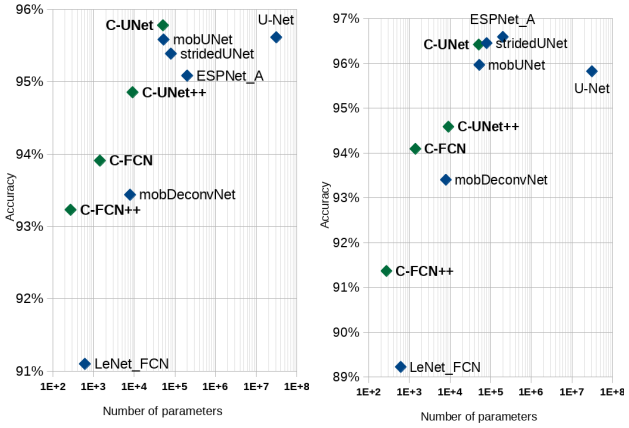


(a) Source image 1 (b) Ground truth (c) U-Net (d) **C-UNet** (e) **C-FCN** (f) **C-FCN++**

(g) Source image 2 (h) Ground truth (i) U-Net (j) **C-UNet** (k) **C-FCN** (l) **C-FCN++**

(m) Source image 3 (n) Ground truth (o) U-Net (p) **C-UNet** (q) **C-FCN** (r) **C-FCN++**

Figure 3: Qualitative results of C-UNet, C-FCN, and C-FCN++ on 38-Cloud dataset, compared to Ground Truth and U-Net [25]. Our networks in bold font. Our networks give good results and are not fooled by snow on the third patch (m). C-FCN and C-FCN++ results appear very smooth compared to others because the segmentation map output by the network is 4 times smaller than U-Net's and upsampled with a bilinear upsampling. Thus, C-UNet produces a more refined segmentation than C-FCN.



(a) 38-Cloud (b) CloudPeru2

Figure 4: Test accuracy with respect to number of parameters on the two tested datasets. Our networks in green and bold font, others in blue. Number of parameters is on a log scale. Our networks offer good accuracy at varying levels of complexity. There is a point of diminishing return in this task: our C-UNet performs about the same as a 500 times bigger U-Net.

Model	Acc.	Prec.	Rec.	Spec.	Jacc.
LeNet_FCN	91.10	86.00	84.34	94.03	74.16
C-FCN++	93.23	91.67	85.45	96.62	79.30
mobDeconvNet [41]	93.44	92.04	85.75	96.78	79.83
C-FCN	93.91	91.23	88.39	96.31	81.47
C-UNet++	94.85	94.18	88.48	97.63	83.89
ESPNet_A [24]	95.08	93.94	89.55	97.49	84.66
StridedUNet [8]	95.39	94.21	90.36	97.58	85.60
mobUNet [41]	95.58	95.37	89.78	98.11	86.03
U-Net [25]	95.61	95.69	89.56	98.25	86.08
C-UNet	**95.78**	**96.53**	89.27	**98.61**	**86.50**
Fmask [43]	94.89	77.71	**97.22**	93.96	75.16
LeNet_FCN	89.23	93.11	83.27	94.52	78.44
C-FCN++	91.37	93.50	87.76	94.58	82.71
mobDeconvNet [41]	93.40	94.98	90.78	95.73	86.63
C-FCN	94.09	95.15	92.15	95.82	88.01
C-UNet++	94.59	96.03	92.31	96.61	88.92
U-Net [25]	95.82	96.29	94.78	96.75	91.44
mobUNet [41]	95.97	**97.20**	94.14	**97.59**	91.66
C-UNet	96.42	96.54	95.83	96.94	92.65
stridedUNet [8]	96.45	96.49	95.95	96.89	92.72
ESPNet_A [24]	**96.59**	96.45	**96.30**	96.85	**93.01**

Table 2: 38-Cloud (top) and CloudPeru2 (bottom) results. Our networks in bold font.

the bilinear upsampling. This is not an issue to estimate cloud coverage as a percentage.

Quantitative results. Table 2 shows accuracy metrics on 38-Cloud and CloudPeru2. We include the Fmask accuracy results from S. Mohajerani *et al*. [26] for 38-Cloud, since our test dataset and metrics are the same.

The best network under 1500 parameters is C-FCN. In particular, this network matches Fmask's accuracy (within 1%), and achieve much better precision and IoU. This network also exceeds the performance of our implementation of mobDeconvNet [41], using 4x less parameters. Our C-FCN++ matches the performance of mobDeconvNet while

using only 273 parameters. Our C-UNet matches the performance of our implementation of mobUNet [41], while using less memory thanks to the lack of skip connections. It also exceeds the performance of our implementation of StridedUNet [8], while being 20% smaller in terms of number of parameters. We see similar results on CloudPeru2 as on 38-Cloud. Our networks provide good results on this high-resolution dataset (3m). Fig. 4 shows the relation between the number of parameters of a network and its accuracy. Our networks match or exceed the performance or other methods at each complexity level.

Impact of skip connections. We tried adding skip connections in C-UNet and C-UNet++, but did not find a significant difference in performance metrics in the case of cloud segmentation, as shown in Table 3. Skip connections are useful to keep high frequency information and use it in the decoder part of the network, since the encoder part tends to act like a low-pass filter because of successive convolutions. Clouds do not have much high frequency information, which is maybe why skip connections were not particularly useful in our case. This means that a significant amount of memory can be saved during network inference by removing skip connections in use cases where they are not useful, as shown in Table 4. This can allow bigger networks to be used on systems with tight resource constraints.

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

Table 3: Impact of skip connections on performance of C-UNet(++) on 38-Cloud dataset. We do not see a significant difference in accuracy when using skip connections for this cloud segmentation task.

Architecture / Image size	384x384	2000x2000
U-Net [25]	34.9 MB	236.5 MB
StridedUNet [8]	4.22 MB	114.4 MB
MobUNet [41]	1.69 MB	45.8 MB
C-UNet with skips	1.97 MB	53.4 MB
C-UNet++ with skips	1.97 MB	53.4 MB
C-UNet without skips	**0 MB**	**0 MB**
C-UNet++ without skips	**0 MB**	**0 MB**

Table 4: Memory footprint of skip connections for 32-bit float inference (in Megabytes), computed for two images sizes.

Performance. Many edge devices use ARM-based SoCs, which are known for their efficiency. This is why we chose the Raspberry Pi, an ARM computer the size of a credit card, for our testing. Our particular board is the Raspberry Pi 4 model B, which features a Quad core Cortex-A72 CPU clocked at 1.5GHz and 2GB of LPDDR4 SDRAM. We feel that this board is representative of the kind of power a typical low-power system could have, as this board consumes 3W when idle and 6W under load on average. We use Tensorflow 1.13.1 with Python 3.7.3 on Raspbian 10.0 to run our evaluation code on the board.

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

Table 5 shows the size used by the tested architectures in terms of number of parameters and storage space, as well as the number of FLOPs (Floating Point Operations) used for one forward pass on a 384x384 image. We can see that the number of FLOPs is indeed correlated to the number of parameters.

Architecture	N_{params}	FLOPs	Storage (MB)
C-FCN++	273	4 654	0.047
LeNet_FCN	614	10 455	0.049
C-FCN	1 438	24 233	0.066
mobDeconvNet [41]	7 919	134 256	0.149
C-UNet++	9 129	154 800	0.172
C-UNet	51 113	867 712	0.735
mobUNet [41]	52 657	893 882	0.724
StridedUNet [8]	79 785	1 355 704	1.0
ESPNet_A [24]	200 193	3 399 310	2.7
U-Net [25]	31 094 497	528 578 588	357

Table 5: Network parameters, FLOPs (FLoating-point OPerationS) and storage size. Our architectures in bold font. FLOPs computed for a 384x384 input using Tensorflow's profiler. We see that the number of FLOPs is correlated to the number of paramters. Storage size is the size of the Keras h5 file for each model in MegaBytes.

Model	384x384	1024x1024	2048x2048
C-FCN++	**0.130**	0.773	**2.663**
stridedUNet [8]	0.133	**0.752**	2.666
C-FCN	0.148	0.903	3.017
C-UNet++	0.204	1.242	4.295
C-UNet	0.404	2.867	10.355
mobUNet [41]	0.688	5.507	OOM
mobDeconvNet [41]	0.755	N/A	N/A
ESPNet_A [24]	1.878	15.940	OOM
U-Net [25]	5.035	OOM	OOM

Table 6: Execution time on images of different sizes in seconds on Raspberry Pi 4, averaged over 20 iterations. OOM = Out of Memory, N/A = mobDeconvNet is not an FCN and thus cannot be executed on images of different sizes than training images.

Table 6 shows the execution time results we obtained. We can see that our networks do not run out of memory when trying to process relatively big images of 2048x2048 pixels at once. Even with a Python 32-bit float implementation, our architectures allow fast processing of images. This processing could be even faster by using Tensorflow Lite and quantized weights.

We notice a strange behaviour with stridedUNet, which performs significantly faster than networks that require less FLOPs. We suspect that Tensorflow, or at least the ARM

version of it, is lacking some optimizations in the case of depthwise separable and atrous convolutions.

Adaptability on FPGA We successfully implemented our most compact network, C-FCN++, on an Altera Cyclone V 5CSXC6 FPGA with a dataflow/streaming architecture and 16-bit quantization based on VGT [10], a tool that automatically generates HDL code for a given neural network architecture. This FPGA will be included aboard OPS-SAT, a CubeSat satellite that will be launched by the end of 2019 by the European Space Agency. Inference on a 2048x1944 image was done in less than 150ms at 100MHz. Thus, C-FCN++ allows real-time processing of images from OPS-SAT's camera as it captures 2000x2000 images at 7 fps. Comparison with the three other networks on FPGA were not possible as they are not compact enough to be implemented on this FPGA.

5. Experiments on forest segmentation

To provide a comparison on another use case, we also evaluate our models on a forest segmentation task, which is also important in remote sensing, for applications such as deforestation monitoring. Although vegetation detection is usually done using dedicated spectral bands on earth observation satellites, it would be useful to be able to perform it on devices that do not usually possess these bands, such as drones or nanosats. Neural networks have already been used successfully for land cover and vegetation segmentation [42, 16]. However, to our knowledge, these tasks have never been done directly onboard.

For this experiment, we use the EOLearn Slovenia 2017 dataset [35], which is composed of multiple Sentinel acquisitions of Slovenia over 2017, with pixel-wise land cover ground truth for 10 classes, as well as cloud masks. We only use the forest class and the RGB bands. This dataset is originally split into 293 1000x1000 patches. We remove images that have more than 10% of clouds and split the remaining images into 500x500 pixel tiles. We use 24 patches as a test set. This means our training/validation set has 12 629 tiles, and our test set has 3 320 tiles. We use the same training scheme and settings as in our cloud segmentation experiments, which we detailed in 4.

In this test, the performance of our networks, as shown in Table 7, is comparable to the other networks, which proves their adaptability to different tasks. In particular, C-UNet matches the performance of U-Net [25], and C-UNet++ matches the performance of mobDeconvNet[41].

Model	Acc.	Prec.	Rec.	Spec.	Jacc.
LeNetFCN	77.10	67.93	63.57	84.22	48.89
C-FCN++	77.40	69.52	61.28	85.87	48.31
C-FCN	78.87	68.95	70.38	83.33	53.44
mobDeconvNet [41]	80.61	71.52	72.68	84.78	56.36
C-UNet++	80.62	72.92	69.61	86.41	55.31
stridedUNet [8]	80.67	72.52	70.70	85.92	55.77
C-UNet	83.33	76.79	73.99	**88.24**	60.47
U-Net [25]	84.04	73.37	**84.28**	83.92	**64.54**
mobUNet [41]	**84.27**	**75.19**	81.13	85.92	63.99

Table 7: Slovenia 2017 forest segmentation results (10m resolution).

6. Conclusion and future works

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

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

ACKNOWLEDGMENT

This work was funded by IRT Saint-Exupéry. We thank the European Space Agency for providing us the FPGA development kit and for supporting our experiment. We would like to thank S. Mohajerani *et al*., G. Morales *et al*. and Sinergise Ltd. for making the 38-Cloud, CloudPeru2 and Slovenia 2017 datasets publicly available.

References

[1] K. Abdelouahab, M. Pelcat, J. Sérot, and F. Berry. Accelerating CNN inference on FPGAs: A survey. *ArXiv*, abs/1806.01683, 2018. 2

[2] V. Badrinarayanan, A. Kendall, and R. Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *PAMI*, 39, 2017. 2

[3] L.-C. Chen, Y. Zhu, G. Papandreou, F. Schroff, and H. Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. In *ECCV*, 2018. 2

[4] F. Chollet. Xception: Deep learning with depthwise separable convolutions. In *CVPR*, 2017. 2

[5] J. Drönner, N. Korfhage, S. Egli, M. Mühling, B. Thies, J. Bendix, B. Freisleben, and B. Seeger. Fast cloud segmentation using convolutional neural networks. *Remote Sensing*, 10(11), 2018. 4

[6] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The Pascal Visual Object Classes (VOC) Challenge. *IJCV*, 88(2), 2010. 2

[7] A. Garcia-Garcia, S. Orts-Escolano, S. Oprea, V. Villena-Martinez, P. Martinez-Gonzalez, and J. Garcia-Rodriguez.

A survey on deep learning techniques for image and video semantic segmentation. *Applied Soft Computing*, 70:41–65, 2018. 2

[8] S. Ghassemi, E. Magli, S. Ghassemi, and E. Magli. Convolutional Neural Networks for On-Board Cloud Screening. *Remote Sensing*, 11(12), 2019. 4, 5, 6, 7

[9] Y. Guo, Y. Liu, T. Georgiou, and M. S. Lew. A review of semantic segmentation using deep neural networks. *International journal of multimedia information retrieval*, 7(2), 2018. 2

[10] M. K. A. Hamdan. *VHDL auto-generation tool for optimized hardware acceleration of convolutional neural networks on FPGA (VGT)*. PhD thesis, Iowa State University, 2018. 2, 7

[11] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 2

[12] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017. 1, 2, 3, 4

[13] M. Hughes, D. Hayes, M. J. Hughes, and D. J. Hayes. Automated Detection of Cloud and Cloud Shadow in Single-Date Landsat Imagery Using Neural Networks and Spatial Post-Processing. *Remote Sensing*, 6(6), 2014. 4

[14] B. Jacob, S. Kligys, B. Chen, M. Zhu, M. Tang, A. Howard, H. Adam, and D. Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *CVPR*, 2018. 2

[15] Jia Deng, Wei Dong, R. Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. ImageNet: A large-scale hierarchical image database. In *CVPR*, 2009. 2

[16] A. Kamilaris and F. X. Prenafeta-Boldú. Deep learning in agriculture: A survey. *Computers and electronics in agriculture*, 147, 2018. 7

[17] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11), 1998. 4

[18] Y. Lecun, J. S. Denker, and S. A. Solla. Optimal Brain Damage. In *NIPS*. 1990. 2

[19] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf. Pruning filters for efficient convnets. *ArXiv:1608.08710*, 2016. 2

[20] D. Lin, S. Talathi, and S. Annapureddy. Fixed point quantization of deep convolutional networks. In *ICML*, 2016. 2

[21] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick. Microsoft COCO: Common objects in context. In *ECCV*, 2014. 2

[22] C.-C. Liu, Y.-C. Zhang, P.-Y. Chen, C.-C. Lai, Y.-H. Chen, J.-H. Cheng, and M.-H. Ko. Clouds classification from sentinel-2 imagery with deep residual learning and semantic image segmentation. *Remote Sensing*, 11(2), 2019. 4

[23] J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *CVPR*, 2015. 1

[24] S. Mehta, M. Rastegari, A. Caspi, L. Shapiro, and H. Hajishirzi. Espnet: Efficient spatial pyramid of dilated convolutions for semantic segmentation. In *ECCV*, 2018. 1, 2, 4, 5, 6

[25] S. Mohajerani, T. A. Krammer, and P. Saeedi. A cloud detection algorithm for remote sensing images using fully convolutional neural networks. In *Workshop on Multimedia Signal Processing*, 2018. 1, 3, 4, 5, 6, 7

[26] S. Mohajerani and P. Saeedi. Cloud-net: An end-to-end cloud detection algorithm for landsat 8 imagery. *ArXiv:1901.10077*, 2019. 4, 5

[27] P. Molchanov, S. Tyree, T. Karras, T. Aila, and J. Kautz. Pruning convolutional neural networks for resource efficient inference. *ArXiv:1611.06440*, 2016. 2

[28] G. Morales, S. G. Huamán, and J. Telles. Cloud detection in high-resolution multispectral satellite imagery using deep learning. In *ICANN*, 2018. 2, 4

[29] R. Mottaghi, X. Chen, X. Liu, N. G. Cho, S. W. Lee, S. Fidler, R. Urtasun, and A. Yuille. The role of context for object detection and semantic segmentation in the wild. In *CVPR*, 2014. 2

[30] H. Noh, S. Hong, and B. Han. Learning deconvolution network for semantic segmentation. In *ICCV*, 2015. 2, 4

[31] O. Ronneberger, P. Fischer, and T. Brox. U-net: Convolutional networks for biomedical image segmentation. In *MICCAI*, 2015. 1, 2, 3, 4

[32] M. Siam, M. Gamal, M. Abdel-Razek, S. Yogamani, and M. Jagersand. RTSeg: Real-Time Semantic Segmentation Comparative Study. *ICIP*, (2), 2018. 2

[33] L. Sifre. *Rigid-motion scattering for image classification*. PhD thesis, 2014. 2

[34] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *ArXiv:1409.1556*, 2014. 2

[35] Sinergise. Modified Copernicus Sentinel data 2017/Sentinel Hub. https://sentinel-hub.com/. 7

[36] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *CVPR*, 2015. 2

[37] S. Wu, G. Li, F. Chen, and L. Shi. Training and inference with integers in deep neural networks. *ArXiv:1802.04680*, 2018. 2

[38] F. Yu and V. Koltun. Multi-scale context aggregation by dilated convolutions. *ArXiv:1511.07122*, 2015. 3

[39] X. Zhang, X. Zhou, M. Lin, and J. Sun. ShuffleNet: An Extremely Efficient Convolutional Neural Network for Mobile Devices. In *CVPR*, 2018. 2

[40] Y. Zhang, B. Guindon, and J. Cihlar. An image transform to characterize and compensate for spatial variations in thin cloud contamination of landsat images. *Remote Sensing of Environment*, 82, 2002. 4

[41] Z. Zhang, G. Xu, and J. Song. Cubesat cloud detection based on JPEG2000 compression and deep learning. *Advances in Mechanical Engineering*, 10, 2018. 4, 5, 6, 7

[42] X. X. Zhu, D. Tuia, L. Mou, G.-S. Xia, L. Zhang, F. Xu, and F. Fraundorfer. Deep learning in remote sensing: A comprehensive review and list of resources. *IEEE Geoscience and Remote Sensing Magazine*, 5(4), 2017. 7

[43] Z. Zhu, S. Wang, and C. E. Woodcock. Improvement and expansion of the fmask algorithm: Cloud, cloud shadow, and snow detection for landsats 4–7, 8, and sentinel 2 images. *Remote Sensing of Environment*, 159, 2015. 4, 5