# FPGA ACCELERATION OF A QUANTIZED NEURAL NETWORK FOR REMOTE-SENSED CLOUD DETECTION

**Philippe Reiter$^{(1)(2)}$, Philipp Karagiannakis$^{(1)}$, Murray Ireland$^{(1)}$, Steve Greenland$^{(1)}$, Louise Crockett$^{(2)}$**

$^{(1)}$*Craft Prospect Ltd, Fairfield, 1048 Govan Road, Glasgow, UK, G51 4XS, Email:phil2@craftprospect.com*
$^{(2)}$*University of Strathclyde, 16 Richmond St, Glasgow, UK, G1 1XQ, Email:philippe.reiter.2019@uni.strath.ac.uk*

## ABSTRACT

The capture and transmission of remote-sensed imagery for Earth observation is both computationally and bandwidth expensive. In the analyses of remote-sensed imagery in the visual band, atmospheric cloud cover can obstruct up to two-thirds of observations, resulting in costly imagery being discarded. Mission objectives and satellite operational details vary; however, assuming a cloud-free observation requirement, a doubling of useful data downlinked with an associated halving of delivery cost is possible through effective cloud detection. A minimal-resource, real-time inference neural network is ideally suited to perform automatic cloud detection, both for pre-processing captured images prior to transmission and preventing unnecessary images being taken by larger payload sensors.

Much of the hardware complexity of modern neural network implementations resides in high-precision floating-point calculation pipelines. In recent years, research has been conducted in identifying quantized, or low-integer precision equivalents to known deep learning models, which do not require the extensive resources of their floating-point, full-precision counterparts. Our work leverages existing research on binary and quantized neural networks to develop a real-time, remote-sensed cloud detection solution using a commodity field-programmable gate array. This follows on developments of the Forwards Looking Imager for predictive cloud detection developed by Craft Prospect, a space engineering practice based in Glasgow, UK.

The synthesized cloud detection accelerator achieved an inference throughput of 358.1 images per second with a maximum power consumption of 2.4 W. This throughput is an order of magnitude faster than alternate algorithmic options for the Forwards Looking Imager at around one third reduction in classification accuracy, and approximately two orders of magnitude faster than the CloudScout deep neural network, deployed with HyperScout 2 on the European Space Agency PhiSat-1 mission. Strategies for incorporating fault tolerance mechanisms are expounded.

## 1. INTRODUCTION

Artificial intelligence (AI), specifically machine learning and deep learning neural networks (NNs) have revolutionized numerous domains of technology during the last decade. From fully autonomous platforms and computer vision to recommender systems and financial forecasting, NNs now form the foundations of numerous computing applications. These algorithms are often heavily resource-consuming, with their applications requiring high-performance computing and datacentre deployments rather than small and low-power devices.

In Earth Observation (EO) analyses and atmospheric cloud detection, real-time inference is required to rapidly identify cloud cover posing an obstruction to visual captures of the planet surface below. With cloud cover possibly enveloping up to two-thirds of the Earth's atmosphere at a given instant, over 60% of remote-sensed imagery may be corrupted by cloud opacity [1]. It ensues that the bandwidth and costs associated with satellite transmissions to ground stations can similarly be reduced by potentially over 60% through the application of an on-board NN for real-time cloud detection. This on-board AI could both prevent obstructed imagery from being transmitted and avoid costly usage of payload instrumentation by anticipating upcoming clouds.

The objective of this work was developing a real-time cloud detector through an NN trained on an EO dataset and implemented on a Commodity Off-The-Shelf (COTS) Field-Programmable Gate Array (FPGA) device. A comparison with existing cloud detection mechanisms is provided in Sec. 2.

### 1.1. Quantization Versus Full Precision

It was once assumed that full-precision calculations were required to obtain accurate results with Deep NNs (DNNs). More recently, researchers have determined that lower-precision, quantized, and even ternary or binary variants of these models can achieve suitable accuracy levels using a fraction of the computing resources. These Quantized NNs (QNNs) can now be implemented using lower power, minimal-resource, embedded System-on-Chips (SoCs) and FPGAs. Sec. 3 captures core learnings, gaps, and opportunities for further innovation from the QNN literature reviewed.

Pattern recognition algorithms implemented using Convolutional NNs (CNNs) are well suited to space exploration and unmanned aerial vehicles and can be used by these applications to identify and classify objects based on captured images [2]. Due to their low-cost, low-power consumption and flexibility, FPGAs offer an attractive solution to efficiently implement NNs

[3]. The performance benefits and flexibility of FPGAs provide a scalable solution and, in recent years, there has been an increased use of these devices in harsh environments, such as space [4].

The end-to-end methodology undertaken for synthesizing a trained cloud detection QNN and implementing it onto an FPGA is covered in Sec. 5.

### 1.2. Implementation Robustness

FPGAs, particularly those based on Static Random-Access Memory (SRAM) technology, have been shown to be sensitive to radiation [5]. They may experience Single Event Upsets (SEU) in the configuration memory that can change the configuration of a routing connection, Look-Up Table (LUT) or Block RAM (BRAM). Therefore, design techniques to mitigate such radiation effects must be applied to these devices [6]. In most cases, the decision on the use of a particular technique will be based on a trade-off between power, area and performance overheads as well as achievable system availability.

Sec. 7 investigates selective hardening [7] mitigation techniques aimed at increasing the robustness of the NN implemented on the FPGA fabric.

### 1.3. AI at the Edge of Outer Space

A summary of results can be referenced in Sec. 6, along with details on the achievement of a low-power, high-throughput solution.

Concluding the paper is a comparison between our obtained results and those of two alternative solutions: The Forwards Looking Imager (FLI) from Craft Prospect, and CloudScout on the European Space Agency (ESA) PhiSat-1 mission.

The FLI uses onboard algorithms for feature detection in the upcoming or near satellite environment, allowing onboard behaviours to be modified to provide a more responsive space asset. The system is highly adaptable to different use-cases for feature detection; however, a common scenario is in providing cloud detection for efficient tasking of a primary payload.

Our accelerator’s throughput advantage is explained, alongside recommendations for improving its accuracy and fault tolerance.

## 2. EO AND CLOUD DETECTION

Remote-sensed imagery for the purposes of cloud detection are typically obtained from satellites in Low-Earth Orbit (LEO) using either visual frequency, infrared, or multi-spectral capture sensors. For the analyses of remote-sensed imagery focused on ground details, cloud cover, even in cases of haze or incomplete opacity, can obstruct the precise study of surface-level topologies, objects and structures. As such, rapidly identifying occurrences of cloud cover, preferably before an image is captured, reduces resource and transmission bandwidth usage along with associated costs. With cost savings often proportional to transmission bandwidth reductions, there are significant gains in time and monetary resources to be gained by avoiding the capture and transmission of unnecessary images containing clouds.

The focus of our research was on visual-band, remote-sensed imagery impacted by clouds. Satellites with synthetic aperture radar (SAR) sensors have wavelengths that can pierce through cloud cover and capture ground data [8], but the reconstructed imagery is of often of lower resolution and targeted for the monitoring of temperature, moisture, and other land metrics rather than visual inspection of land details.

Multi-spectral sensors fitted on an increasing number of satellites can provide upper-atmosphere moisture and temperature readings, which can be used in the differentiation of clouds from ground and sea effects with similar spectra – such as ice, snow, or white-coloured, human-made objects [1][8]. Since remote-sensed captures using Red, Green, Blue (RGB) and near-infrared sensors remain a prominent use-case with a number of properly annotated datasets, the visual band was the focus for this research – similar to [1] and [9]. For these latter research undertakings, the infrared channel was also considered, whereas this was outside the scope of the strictly RGB observations used in our research.

### 2.1. Conventional Cloud Detection Techniques

A conventional method of cloud detection with RGB imagery is to perform per-pixel analyses. This involves marking each pixel as either cloud-containing or not using classical machine learning algorithms, such as decision trees or support vector machines [10]. These processes, which include Function of Mask and Automated Cloud Cover Assessment, rely on a specific transmission band from a satellite to inform the post-processing analysis of the intensity spectra required to make the cloud / no-cloud decision. While being lightweight detection algorithms, research has shown these approaches to be prone to erroneously classifying snow and ice cover, or white-coloured land or sea objects as clouds [8].

In lieu of the manual specification of features for clouds, research has been conducted in devising deep learning solutions for the task [8][11][12]. A primary advantage of DNNs over classical machine learning functions and shallow Artificial NNs (ANNs) is their ability to automatically derive detection features of interest as part of their training and optimization [9]. A couple of DNN approaches that informed the network architecture used for this research are examined in the following subsection.

### 2.2. Deep Learning for Cloud Detection

Neural networks require training on sufficiently large datasets to calibrate their weights between neural nodes. The objective of training is to minimize the loss in prediction or classification against either manually or automatically generated expected results. Deep models compared to their ANN counterparts have multiple layers of neurons and increasingly feature novel functionality, including the convolutional layers and non-linear activation functions found in CNNs.

When the data inputted into a learning model is pre-annotated, the training process is said to be supervised. Networks requiring no prior labelling of data are trained in an unsupervised manner. A semi-supervised intermediate case exists for specific training instances. A supervised approach was undertaken for our research.

### 2.3. Ground Truths

To perform loss minimization during supervised DNN training, the backpropagation process requires expected prediction or classification results. These ground truths for the data are the labels, or annotations previously mentioned.

Per-pixel cloud masks were employed by [1] and [9] to gauge the accuracy of cloud detections. These cloud masks represented the presence or absence of cloud cover in a pixel using binary values – 1 for some cloud, or 0 otherwise. The satellite data used by this former research permitted the availability or generation of these per-pixel prediction maps.

As the datasets used for this research included per-image cloud cover classifications, and not per-pixel granularity, the NNs investigated were required to learn from each image what features constituted clouds or ground topologies. This approach reduced the computational requirements for the network, as detection was performed per 32x32 tile of atmospheric sky capture and categorized into one of initially three classes.

Per-pixel and per-region, or “superpixel” [9], detection accuracies are needed for DNNs featuring image segmentation capabilities. These networks are capable of individually identifying areas of an image with or without cloud cover. While such networks with added architectural complexity were investigated, they were not required for our cloud detection use-case and are mentioned for further study.

## 3. NEURAL NETWORK ARCHITECTURE

The rapid rise of interest by research and industry for DNNs in the last decade was predicated on the relatively low-cost availability of high-performance matrix computation hardware. Specifically, the deep, parallel pipeline architectures of Graphics Processing Units (GPUs), with their specialized matrix and vector multiplication functions, resulted in the almost-exclusive use of GPUs as accelerators in the advancement of DNNs since the pivotal publication of the AlexNet framework by Krizhevsky, Sutskever and Hinton in 2012. AlexNet was trained with the ImageNet dataset of over one million annotated images [13] using

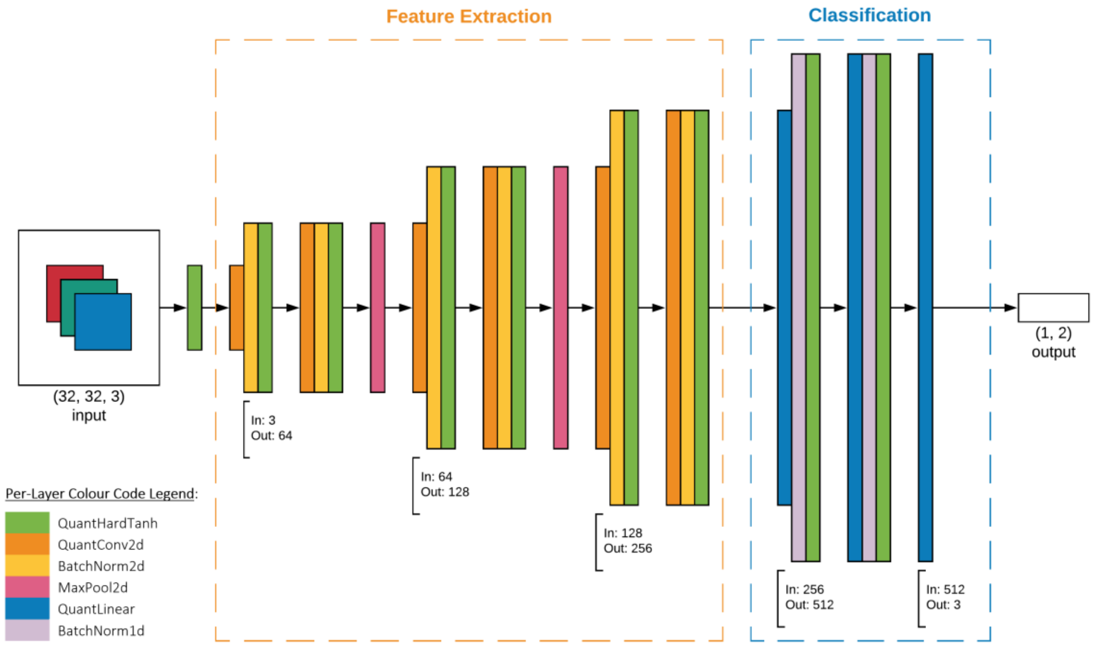

*Figure 1. CNV architectural layer diagram illustrating the progression of data from image input into convolutional feature extraction and through linear fully-connected classification to produce a two-class prediction vector. The dimension of features entering a convolutional or fully-connected layer is specified as the “In” value, while the resulting dimension is labelled “Out.”*

two Nvidia GTX 580 GPUs [14]. The astounding improvement in ImageNet classification by AlexNet, demonstrating an over ten-percent reduction in classification loss compared to its closest NN competitor, spotlighted the use of off-the-shelf GPU hardware for DNN training and inference acceleration. These devices had previously been targeted towards computer image rendering and video gaming, but were now aptly positioned to accelerate research in AI and compute applications in the datacentre.

The Multiply-Accumulate (MAC) units on GPUs rapidly perform high-precision, floating-point matrix and vector computations, and each unit can be regarded as an artificial neuron in a DNN [15].

### 3.1. Binarized and Quantized Neural Networks

CNNs require the storage and processing of thousands to millions of parameters, learned through numerous epochs of training on large datasets. These dictate the convolutional behaviour that leads to accurate inferences of a variety of classification types. Research into CNN kernels, or filters, have determined that a significant percentage of the convolutional kernel parameters are duplicated, and there is efficiency to be gained in sparse matrix layouts for CNNs [15]. Moreover, novel NN architectures, using smaller but deeper convolutional layers, such as SqueezeNet, have demonstrated close to parity with AlexNet performance, but with over five-hundred times fewer parameters [16].

The precision of each parameter contributes to the computational load of a network. Even reduced architectures store hundreds of thousands of weights and activation parameters, typically with 16- or 32-bit Floating Point (FP) precision. Quantization of weight and activation values can be employed to reduce parameter precision while maintaining a consistent NN architecture. Common quantization levels include 4- and 8-bit fixed-integer (INT4 and INT8, respectively), as well as ternary (2-bit) and binary (1-bit) representations. Full-precision values can also be gradually quantized through a series of thresholding steps [17].

By quantizing down to the binary values of 1 or -1, Courbariaux et al created a Binary NN (BNN). In their BNN, weight values above or equal to zero are set to 1, while all negative weights are set to -1. This single-bit, switch-like implementation decreases storage requirements and reduces hardware demands [15].

Given the binary nature of the weights, the multiplications performed by high-precision MAC units on GPUs can be completed using simple XNOR gates or bitwise shifts. Furthermore, accumulation operations can be performed through bit count functions that are fundamental to most microprocessors. Therefore, a 32-bit FP MAC unit, which can consume hundreds of FPGA slices, can be replaced with a single or two-slice XNOR-plus-bit count structure. This reduction in computing resources also leads to a corresponding decrease in the number of required memory accesses and overall power consumption of the implementation. Moreover, through the definition of a custom instruction set on an FPGA, a specific XNOR-plus-bit count instruction can be created. This special instruction has the advantage of computing in a single passthrough what would typically consume upwards of a MAC cycle on a GPU. Further optimizing the design, the 32-bit registers previously used to store full-precision weights can be repurposed for 32, one-bit operations in Single Instruction, Multiple Data (SIMD) execution [15].

### 3.2. Convolutional BNN

The BNN selected for our research was based on a classical CNN layout and used the 1-bit weight and activation, CNV-W1A1, fully convolutional CNV architecture from the open-source Xilinx Brevitas libraries [18]. Fig. 1 diagrams the feature extraction and classification stages of the CNV architecture. The input dimensions, (32, 32, 3) specified correspond to 32x32 RGB, or 3-channel images.

The feature extraction stage of the network consists of a sequence of convolution functional blocks. Each grouping of functions consists of a 2D quantized

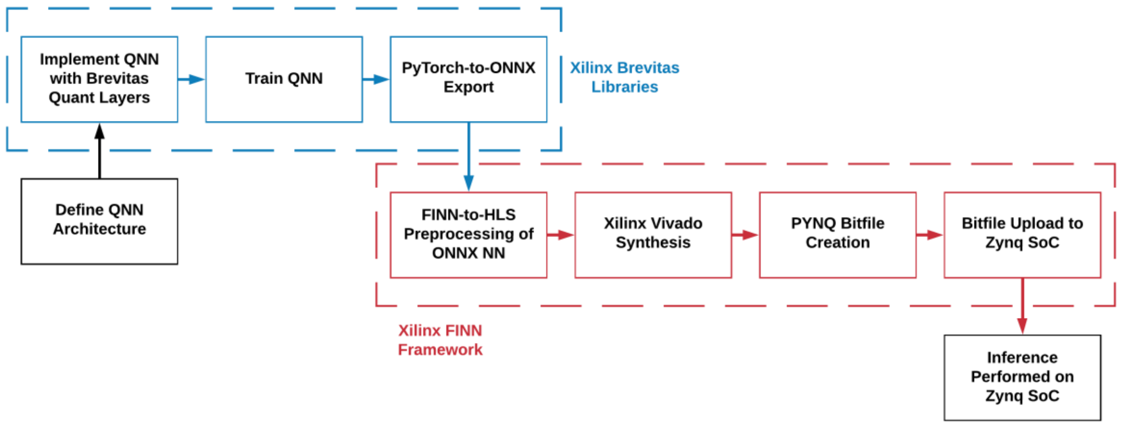
Implement QNN with Brevitas Quant Layers
Train QNN
PyTorch-to-ONNX Export
Xilinx Brevitas Libraries
Define QNN Architecture
FINN-to-HLS Preprocessing of ONNX NN
Xilinx Vivado Synthesis
PYNQ Bitfile Creation
Bitfile Upload to Zynq SoC
Xilinx FINN Framework
Inference Performed on Zynq SoC

*Figure 2. High-level diagram depicting Brevitas export to the FINN framework followed by ONNX graph processing, synthesis, driver creation, and board deployment. Diagram inspired from [17].*

convolution operation, QuantConv2d, which applies a 3x3 filter to its input with a stride of 1. The dimensions of the resulting feature matrix from a QuantConv2d layer is either maintained or doubled every other convolution. According to research conducted by [19], performing batch normalization following a convolution, and prior to applying an activation function, is preferable for QNNs. As such, each QuantConv2d function is immediately followed by a 2D batch normalization, BatchNorm2d, that then outputs to a hard tanh activation function, QuantHardTanh.

Max pooling layers, MaxPool2d, are inserted as downsamplers to restrain the number of features between convolutional groupings and introduce noise. Average pooling is not supported for binarized networks [19], so this layer type was not considered.

Feature classification is performed through linear, fully-connected layers, QuantLinear, in the latter stage of the CNV architecture. Similar to the convolution functional groupings, fully-connected groupings comprise of a QuantLinear layer followed by vector batch normalization and the same QuantHardTanh activation used for convolutions. At the output of the classification stage, a final QuantLinear layer compresses a vector of 512 features to one of two prediction values corresponding to the two classification labels on which the final network was trained.

## 4. NEURAL NETWORK SYNTHESIS

Once trained, a BNN or QNN can be translated from software language implementation to hardware modules through specialized mapping and synthesis. The open-source Xilinx FINN framework was applied for this project to facilitate QNN-to-hardware mapping, synthesis and deployment.

FINN accepts networks in ONNX format and assigns each layer of a QNN to a compute engine. A high degree of parallelism is applied between and within each compute engine of a QNN. These compute units consist of Processing Elements (PEs) operating in parallel, as well as register-level parallelism through SIMD instructions.

NN downscaling strategies are also employed by FINN, including weight reduction through redundancy elimination and thresholding; sparse matrix calculations; and, fixed-integer, ternary and binary quantization [19]. It applies the custom instruction and parallelism optimisations described in [15] to the translation of a QNN to an efficient FPGA implementation.

In order to minimize memory access latencies, QNN weight and activation values are closely coupled with associated logic elements through FPGA BRAM and LUT RAM [17]. A high-level diagram of the FINN end-to-end flow is diagramed in Fig. 2.

## 5. METHODOLOGY

The datasets, and hardware and software tools used to design, implement, train and test BNN and QNNs on an FPGA platform are detailed in the following subsections.

### 5.1. Development Platform

The target development board for this research was a Diligent Arty Z7-20 with Xilinx Zynq-7000 SoC. The SoC, an XC7Z020-1CLG400C device, is equipped with dual ARM Cortex-A9 Central Processing Unit (CPU) cores operating at 650 MHz, coupled with a Xilinx Artix-7 FPGA. It features 512 MB of DDR3 off-chip memory, and 630 KB of FPGA BRAM.

For on-board operations, the open-source PYNQ software platform [20] from Xilinx supports the Python programming language and the interactive Python notebook application, Jupyter. PYNQ-Z1 version 2.5 was loaded onto the Arty-Z7 board for this research.

### 5.2. Frameworks and Libraries

The BNN and QNNs trained and tested were implemented using version 1.1.0 of the PyTorch framework and version 0.2.0 of the Xilinx Brevitas libraries. Network synthesis was performed using Xilinx FINN version 0.3b and the Xilinx Vivado 2019.1 high-level synthesis tool.

### 5.3. Datasets

The datasets used for training were a custom ESA Copernicus Sentinel-2 image-set, in JPEG format, provided by Craft Prospect (CPL), and the “Planet: Understanding the Amazon from Space” dataset obtained via Kaggle [21].

The Sentinel-2 cloud detection image-set comprised a total of 30,918 images post rebalancing, which were initially categorized into three classes – cloudy, partly cloudy, and clear. Each sample had a resolution of 47x47 pixels with RGB colour channels (47x47x3). During the latter phase of our research, the classification scheme was updated to a binary use-case – cloudy, which included partly cloudy, and clear.

Planet’s dataset contained 24,094 images, following rebalancing between clear and cloud-containing classes, with a per-sample resolution of 256x256x3. These images were similarly annotated to those from the Sentinel-2 set.

To test for greater network generalization, a combined Sentinel-2 and Planet dataset, containing a total of 52,315 images, was created.

A 70:20:10 training, validation, and test split was selected for all datasets. This ensured a sufficient number of training and validation samples. No image augmentation was performed; image scaling and normalization were the only transforms applied.

## 5.4. Performance Benchmark

The FLI was developed by Craft Prospect using a Zynq-7000 SoC similar to that used for this research. According to [11], the device can achieve an inference throughput of 5.3 images per second (images/sec). The training accuracy for the deep learning model is stated as 98%, with a cloud-cover inference accuracy of 93%. A duty cycle power consumption of 0.9 W is quoted, with a maximum instantaneous power consumption of approximately 2.2 W.

While the FLI is the most directly comparable accelerator to that implemented during this research, the ESA PhiSat-1 mission, which launched on September 3rd, 2020, includes the HyperScout 2 multi-spectral vision module with CloudScout cloud detection accelerator [22][23]. The objective of the PhiSat-1 endeavour is to demonstrate the advantages of using DNN technology to increase the efficiency and effectiveness of Earth observations. As such, the PhiSat-1 mission is of significant interest when considering the applicability of our research to future satellite missions.

The CloudScout DNN is implemented on an Intel Movidius Myriad 2 Vector Processing Unit (VPU). Unlike the FPGA logic on the Zynq-7000 SoC, which can be reprogrammed to implement a custom NN architecture, the VPU is an Application-Specific Integrated Circuit (ASIC) tailored to accelerate general NN applications.

While the FPGA accelerator implemented for our research architecturally differed from CloudScout, the latter provided a current baseline on which to compare performance, power consumption and accuracy. Based on the results in [23], CloudScout has an inference throughput of 3.1 images/sec, consuming 1.8 W of power per inference cycle. An on-board test accuracy of 92% is quoted.

## 6. RESULTS

After experimenting with varying batch sizes, learning rates, optimizers and loss functions, the hyperparameter configuration that yielded the best inference results for the CNV-W1A1 model employed an Adam optimizer, with an initial learning rate of 0.001 and momentum of 0.0001, along with the multi-margin loss function from PyTorch. The latter was used in lieu of a hinge loss, which was found to be effective in prior BNN research [11]. The best validation accuracy achieved for CNV-W1A1 was 91.9% using a batch size of 50 and trained for 190 epochs.

Training and inference results were obtained using a binary classification use-case of either clear or cloudy – the latter of which encompassed both partly cloudy and over-90% obscured images. Since no current mission requirement for the network generalization afforded from a combined, cross-geography dataset was found, training and inference trials used solely Sentinel-2 images without the set from Planet.

## 6.1. Inference and Throughput

Inference comparisons between CPU (Intel Xeon E5-2686 v4, 2.30GHz), GPU (Nvidia Tesla M60, 8GB GDDR5) and FPGA (Xilinx Zynq-7000 XC7Z020-1CLG400C) implementations demonstrated a marked accuracy difference, as listed in Tab. 1. While both the CPU and GPU achieved over 91% accuracy on the Sentinel-2 test set, the FPGA scored 64.0%. This disparity in results is primarily attributed to the quantization pre-processing required to load the image data onto the FPGA, but which was not similarly performed for software-based inference.

*Table 1. Inference accuracies using the Sentinel-2 test set and FPGA, GPU and CPU accelerators.*

| **Inference Accuracy** | **Total (%)** | **Clear (%)** | **Cloudy (%)** |
|---|---|---|---|
| **FPGA** | 64.0 | 74.6 | 53.4 |
| **GPU** | 91.9 | 93.0 | 95.4 |
| **CPU** | 91.8 | 90.7 | 85.3 |

The FPGA accelerator excelled in inference throughput, processing 358.1, 32x32 images per second using a maximum power consumption of 2.4 W at a 100 MHz target frequency.

*Table 2. Runtime and throughput performance for each of the accelerator platforms.*

| | **Throughput (images/sec)** | **Runtime (millisec)** |
|---|---|---|
| **FPGA** | 358.1 | 2.8 |
| **GPU** | 45.5 | 22.0 |
| **CPU** | 35.6 | 28.1 |

For comparison, as shown in Tab. 2, the throughput performance achieved by the synthesized CNV-W1A1 was 7.9-times faster and consumed approximately 120 times less power [24] than what was recorded using the GPU, which is the conventional datacentre inference accelerator of choice.

## 6.2. Resource Utilization

The CNV-W1A1 implementation consumed a little over half of the available LUTs, 36% of the flip-flops (FFs), and 98.6% of the BRAM on the FPGA, as summarized in Tab 3.

To ensure minimal inference latency, the FINN synthesis process constrained the network to the on-chip BRAM and LUTRAM. Off-chip DRAM memory accesses had a bandwidth of only 1.1 Mb/s inbound and 4.3 kb/s outbound, so storing all network parameters in FPGA memory was advantageous.

*Table 3. Resource utilization summary (from Vivado 2019.1) for the CNV-W1A1 network on the FPGA.*

| Resource | Utilization (%) |
|---|---|
| **LUT** | 52.1 |
| **LUTRAM** | 21.8 |
| **FF** | 33.9 |
| **BRAM** | 98.6 |
| **BUFG** | 3.1 |

## 7. FAULT TOLERANCE AND HARDENING

The preceding results have demonstrated the effectiveness of an FPGA for implementing an NN. However, these types of programmable devices are susceptible to radiation effects and require special fault mitigation techniques targeting the configuration memory, user logic, and embedded RAM blocks. The decision on the use of a particular technique is based on a trade-off between power, area and performance overheads as well as achievable system availability. This section focuses on mitigation techniques specifically aimed at increasing the robustness of the NN implemented on the FPGA fabric, as well as the results that implementing such techniques have on power consumption and resource availability.

Triple Modular Redundancy (TMR) is one of the most common and effective methods of implementing SEU mitigation in an SRAM-based FPGA. It can be implemented at different granularities – register level, block level, or device level –, and involves triplication of a design portion with added voting logic to filter out incorrect results in one of the three signal paths [25][26][27]. Although full TMR leads to the best possible system for mitigating effects of radiation, the increased resource usage has a direct impact on power consumption and heat generation – both unwanted in the context of satellite systems. If a design is too large to apply full TMR, alternatives include partial TMR and Reduced Precision Redundancy.

Understanding the impact of radiation on the outcomes of an NN offers an opportunity to customise TMR-based mitigation techniques to the implemented model. Libano et al showed that radiation can induce errors that modify the output of the network with or without affecting the NN's functionality [7]. Discerning if an error had a critical effect on the NN functionality permitted the identification of its most vulnerable layers, along with a selective hardening strategy that triplicated only those layers likely to influence the outcome. Applying this technique, Libano et al were able to achieve almost 70% fault masking with a 45% increase in resource utilisation. This is an impressive performance gain when compared to a full TMR solution, which would outperform the selective configuration in terms of reliability, but would require a 200% increase in area to reach a nearly 100% rate of fault masking.

As listed in Tab. 3, with 98.6% of the FPGA BRAM used for the CNV-W1A1 implementation, there was no opportunity to apply even partial TMR to improve the output of the network. Possible solutions include using a larger device or trimming the network to free resources and ensure that it is robust to radiation. Further work is needed to assess these techniques for the implemented network and their effects and efficiency at mitigating faults.

## 8. CONCLUSION

The binarized CNN, CNV-W1A1, implemented on a Xilinx Zynq-7000 SoC demonstrated the rapid cloud detection throughput possible from using a custom-synthesized NN logic on an FPGA accelerator. An inference throughput of 358.1 images/sec was achieved using the implemented CNV-W1A1 accelerator. This throughput is over 162 times faster than the minimum 2.2 images/sec required for effective cloud detection, as specified in [11]. Moreover, the throughput of the CNV-W1A1 deployment was nearly 68 times greater than the FLI, with 5.3 images/sec. It was also 115 times faster than the in-orbit CloudScout DNN with 3.1 images/sec. The CNV-W1A1 throughput exceeded that of even a high-performance GPU accelerator by completing the same inference testing in 7.9-fold less time.

The impressive throughput of the CNV-W1A1 implementation was contrasted with an inference accuracy lagging that of the FLI and CloudScout – 64.0% compared to 93.0% and 92.0%, respectively. The CloudScout implementation, however, used FP16 precision for its network, compared to the binary weights and activations of the CNV-W1A1 model.

With increasingly more stringent and competitive restrictions on the footprint, power consumption and heat dissipation of orbital space technology, greater efficiency is required to both augment the value of results and decrease associated operational costs. LEO missions equipped with embedded AI devices are demonstrating the cost-savings, performance boosts, and end-result advantages afforded by NN-powered automation and on-board deep learning in outer space.

Deployments in this domain are still nascent, with extensive potential for AI and deep learning in enumerable facets of space technology. The critical factors remain those of maintaining a compact, performant, fault-tolerant and accurate solution that reliably functions within a low-power envelope.

Our research has demonstrated the inference throughput advantage of an FPGA accelerator for real-time cloud detection using a synthesized BNN. This work will go on to feed further enhancements and development of the Craft Prospect FLI. Such custom-configured FPGA implementations showcase the potential for low-power, high-throughput AI accelerators not only at the edge of global networks, but that of outer space.

## REFERENCES

[1] Mohajerani, Sorour, Thomas A. Krammer, and Parvaneh Saeedi. "A Cloud Detection Algorithm for Remote Sensing Images Using Fully Convolutional Neural Networks." In *2018 IEEE 20th International Workshop on Multimedia Signal Processing (MMSP)*, pp. 1-5. IEEE, 2018.

[2] Campbell, Tanner S. "A Deep Learning Approach to Autonomous Relative Terrain Navigation." PhD diss., The University of Arizona, 2017.

[3] Lentaris, George, Konstantinos Maragos, Ioannis Stratakos, Lazaros Papadopoulos, Odysseas Papanikolaou, Dimitrios Soudris, Manolis Lourakis, Xenophon Zabulis, David Gonzalez-Arjona, and Gianluca Furano. "High-performance embedded computing in space: Evaluation of platforms for vision-based navigation." *Journal of Aerospace Information Systems* 15, no. 4 (2018): 178-192.

[4] Caffrey, Michael, Manuel Echave, Charles Fite, Tony Nelson, Anthony Salazar, and Steven Storms. "A space-based reconfigurable radio." In *Proceedings of the international conference on engineering of reconfigurable systems and algorithms (ERSA)*, pp. 49-53. CSREA Press, 2002.

[5] Wirthlin, Michael. "High-reliability FPGA-based systems: space, high-energy physics, and beyond." *Proceedings of the IEEE* 103, no. 3 (2015): 379-389.

[6] Siegle, Felix, Tanya Vladimirova, Jørgen Ilstad, and Omar Emam. "Mitigation of radiation effects in SRAM-based FPGAs for space applications." *ACM Computing Surveys (CSUR)* 47, no. 2 (2015): 1-34.

[7] Libano, F., B. Wilson, J. Anderson, M. J. Wirthlin, C. Cazzaniga, C. Frost, and P. Rech. "Selective hardening for neural networks in FPGAs." *IEEE Transactions on Nuclear Science* 66, no. 1 (2018): 216-222.

[8] Xie, Fengying, Mengyun Shi, Zhenwei Shi, Jihao Yin, and Danpei Zhao. "Multilevel cloud detection in remote sensing images based on deep learning." *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing* 10, no. 8 (2017): 3631-3640.

[9] Le Goff, M., J-Y. Tourneret, H. Wendt, M. Ortner, and M. Spigai. "Deep learning for cloud detection." In *8th International Conference of Pattern Recognition Systems (ICPRS 2017)*, pp. 1-6. IET, 2017.

[10] Latry, Ch, Ch Panem, and Ph Dejean. "Cloud detection with SVM technique." In *2007 IEEE International Geoscience and Remote Sensing Symposium*, pp. 448-451. IEEE, 2007.

[11] Greenland, Steve, Murray Ireland, Chisato Kobayashi, Peter Mendham, Mark Post, and David White. "Development of a minaturised forwards looking imager using deep learning for responsive operations." In *4S Symposium 2018: The Symposium on Small Satellites for Earth Observation*. 2018.

[12] Dev, Soumyabrata, Atul Nautiyal, Yee Hui Lee, and Stefan Winkler. "Cloudsegnet: A deep network for nychthemeron cloud image segmentation." *IEEE Geoscience and Remote Sensing Letters* 16, no. 12 (2019): 1814-1818.

[13] Deng, Jia, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In *2009 IEEE conference on computer vision and pattern recognition*, pp. 248-255. IEEE, 2009.

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

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

[16] Iandola, Forrest N., Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and < 0.5 MB model size." *arXiv preprint arXiv:1602.07360* (2016).

[17] Umuroglu, Yaman, Nicholas J. Fraser, Giulio Gambardella, Michaela Blott, Philip Leong, Magnus Jahre, and Kees Vissers. "FINN: A framework for fast, scalable binarized neural network inference." In *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 65-74. 2017.

[18] Xilinx Research Labs. "Brevitas: quantization-aware training in Pytorch." Retrieved May 26, 2020 from <https://github.com/Xilinx/brevitas>.

[19] Blott, Michaela, Thomas B. Preußer, Nicholas J. Fraser, Giulio Gambardella, Kenneth O'brien, Yaman Umuroglu, Miriam Leeser, and Kees Vissers. "FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks." *ACM Transactions on Reconfigurable Technology and Systems (TRETS)* 11, no. 3 (2018): 1-23.

[20] Xilinx. "PYNQ: Python Productivity for ZYNQ." Retrieved May 18, 2020 from <http://www.pynq.io>.

[21] Planet (2017). "Planet: Understanding the Amazon from Space." Retrieved from Kaggle on May 26 2020 from <https://www.kaggle.com/c/planet-understanding-the-amazon-from-space>.

[22] eoPortal Directory (2020). "PhiSat-1 Nanosatellite Mission." Retrieved August 18, 2020 from: <https://directory.eoportal.org/web/eoportal/satellite-missions/p/phisat-1>.

[23] Giuffrida, Gianluca, Lorenzo Diana, Francesco de Gioia, Gionata Benelli, Gabriele Meoni, Massimiliano Donati, and Luca Fanucci. "CloudScout: A Deep Neural Network for On-Board Cloud Detection on Hyperspectral Images." *Remote Sensing* 12, no. 14 (2020): 2205.

[24] Nvidia, "NVIDIA® Tesla® M60 GPU Accelerator" datasheet. Retrieved August 18, 2020 from <https://images.nvidia.com/content/tesla/pdf/188417-Tesla-M60-DS-A4-fnl-Web.pdf>.

[25] Pratt, Brian, Michael Caffrey, James F. Carroll, Paul Graham, Keith Morgan, and Michael Wirthlin. "Fine-grain SEU mitigation for FPGAs using partial TMR." *IEEE Transactions on Nuclear Science* 55, no. 4 (2008): 2274-2280.

[26] Kastensmidt, F. Lima, Luca Sterpone, Luigi Carro, and M. Sonza Reorda. "On the optimal design of triple modular redundancy logic for SRAM-based FPGAs." In *Design, Automation and Test in Europe*, pp. 1290-1295. IEEE, 2005.

[27] Bolchini, Cristiana, Antonio Miele, and Marco D. Santambrogio. "TMR and Partial Dynamic Reconfiguration to mitigate SEU faults in FPGAs." In *22nd IEEE International Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT 2007)*, pp. 87-95. IEEE, 2007.