Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

*Digital Object Identifier 10.1109/ACCESS.2019.DOI*

# Low-Power and High-Speed Deep FPGA Inference Engines for Weed Classification at the Edge

**COREY LAMMIE[1], (Student Member, IEEE), ALEX OLSEN[1], (Student Member, IEEE), TONY CARRICK[1], (Student Member, IEEE), and MOSTAFA RAHIMI AZGHADI[1], (Member, IEEE)**

[1]Neural-inspired Computing and Engineering (NICE) Lab, College of Science and Engineering, James Cook University, Townsville, QLD 4814, Australia

Corresponding authors: Corey Lammie and Mostafa Rahimi Azghadi (corey.lammie@jcu.edu.au, mostafa.rahimiazghadi@jcu.edu.au)

**ABSTRACT** Deep Neural Networks (DNNs) have recently achieved remarkable performance in a myriad of applications, ranging from image recognition to language processing. Training such networks on Graphics Processing Units (GPUs) currently offers unmatched levels of performance; however, GPUs are subject to large power requirements. With recent advancements in High Level Synthesis (HLS) techniques, new methods for accelerating deep networks using Field Programmable Gate Arrays (FPGAs) are emerging. FPGA-based DNNs present substantial advantages in energy efficiency over conventional CPU- and GPU-accelerated networks. Using the Intel FPGA Software Development Kit (SDK) for OpenCL development environment, networks described using the high-level OpenCL framework can be accelerated targeting heterogeneous platforms including CPUs, GPUs, and FPGAs. These networks, if properly customized on GPUs and FPGAs, can be ideal candidates for learning and inference in resource-constrained portable devices such as robots and the Internet of Things (IoT) edge devices, where power is limited and performance is critical. Here, we introduce GPU- and FPGA-accelerated deterministically binarized DNNs, tailored toward weed species classification for robotic weed control. Our developed networks are trained and benchmarked using a publicly available weed species dataset, named DeepWeeds, which includes close to 18,000 weed images. We demonstrate that our FPGA-accelerated binarized networks significantly outperform their GPU-accelerated counterparts, achieving a $>$7-fold decrease in power consumption, while performing inference on weed images 2.86 times faster compared to our best performing baseline full-precision GPU implementation. These significant benefits are gained whilst losing only 1.17% of validation accuracy. This is a significant step toward enabling deep inference and learning on IoT edge devices, and smart portable machines such as an agricultural robot, which is the target application in this paper.

**INDEX TERMS** Machine Learning (ML), Deep Neural Networks (DNNs), Convolutional Neural Networks (CNNs), Binarized Neural Networks (BNNs), Internet of Things (IoT), Field Programmable Gate Arrays (FPGAs), High-level Synthesis (HLS), Weed Classification.

## I. INTRODUCTION

THE promise of robotic weed control to provide a step change in the productivity of primary industry is widely coveted [1], [2]. The rationale is clear - replace human involvement in this time-consuming and labor-intensive undertaking with more efficient autonomous machines. In addition, improving the efficacy of weed control would have enormous economic impact [3]. The majority of current works in this arena focus on the four core technologies of robotic weed control: detection, mapping, guidance, and control [4]. The robust and efficient detection of weed species remains a major obstacle to the widespread uptake of robotic weed control technologies [3].

The use of DNNs specifically tasked for plant classification has demonstrated incredible performance in recent works [5]–[7]. Using the Intel FPGA SDK for OpenCL development environment, networks described using the high-level OpenCL framework can be accelerated by targeting heterogeneous platforms with CPUs, GPUs, and FPGAs. These developed networks are ideal candidates for edge computing applications, where low-power consumption and high performance are critical.

In this work, we investigate the acceleration of binarized DNNs on GPUs and FPGAs using the high-level OpenCL framework for weed species classification targeted toward robotic weed control, as depicted in Figure (1). We demonstrate that our FPGA implementations, employing an *Intel DE1-SoC* FPGA development board, customized for edge processing, exhibit comparable performance to state-of-the-art hardware implementations, employing an *NVIDIA Titan V* GPU and an *AMD Ryzen 2700X @ 4.10 GHz Overclocked (OC)* CPU, which are typically used for conventional desktop-based processing. Our specific contributions are five-fold:

- We implement and present the first FPGA-accelerated binarized DNN specifically tasked for weed species classification.
- Our complete FPGA-accelerated DNN runs on a standalone System On a Chip (SoC), requiring no host computer or extra device for partial computation.
- We investigate the effect of down-sampling input images on the DNN classification accuracy, and demonstrate that significantly reducing the image resolution has a marginal effect on accuracy.
- We thoroughly compare our efficient implementations on GPU and FPGA platforms and demonstrate that our new binarized FPGA-accelerated DNNs offer significantly reduced power usage while lowering per-image inference times compared to their conventional GPU-accelerated counterparts.
- We make our software code publicly available, to provide the community with the opportunity to replicate our experimental results and to adapt our utilized techniques for their various applications.

The paper is structured as follows: Section II presents related works. Section III presents an overview of the algorithms and methods used in our designed networks. Section IV presents our new labeled version of the DeepWeeds dataset [8], *DeepWeedsX*. Our image pre-processing techniques used are detailed in Section V. Section VI presents our developed software and hardware network architectures. Section VII reports the effect of image down-sampling on the network performance. Section VIII presents and discusses our software and hardware results, while Section IX provides further discussions on classification and real-time performance of the proposed design. The paper is concluded in Section X.

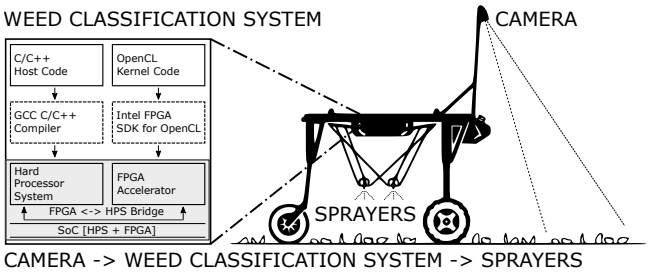


**FIGURE 1:** A block diagram detailing a complete robotic weed management system, where our developed weed classification inference engine is used for low-power and real-time weed classification, that can trigger herbicide sprayers for the detected and classified weed species.

## II. RELATED WORKS

A variety of techniques have been explored to automatically detect and classify target plant life. Means of detection can be categorised into one of three representations of the light spectrum: image-based [5]–[7], [9]–[15], spectrum-based [16], [17] and spectral image-based [18], [19].

In [8], we have achieved and demonstrated 95.7% weed classification accuracy on our large multiclass weed species image dataset, named DeepWeeds, using the ResNet-50 [20] Convolutional Neural Network (CNN) architecture on a single NVIDIA GTX 1080Ti GPU.

Performing real-time learning and inference, targeted for plant classification, on high-performance GPUs such as the NVIDIA GTX 1080Ti, is putative to consume large amounts of power, and hence, is ill-suited to deployment in portable resource-constrained smart devices and robotic systems, which are becoming commonplace. Consequently, devising methods and hardware synthesis techniques for reducing power consumption and for improving throughput of DNN hardware, becomes formidable.

While GPU-based implementations targeted towards image classification tasks are plentiful [21]–[26], only one recent work [27] has detailed the implementation of custom hardware accelerators specifically tasked for agricultural purposes. [27] demonstrates real-time performance by implementing an FPGA-based DNN on a Terasic DE1-SoC for plant detection in organic farming. The system developed in [27] can classify a target dataset with accuracy matching that of a state-of-the-art GPU while running at up to 42 frames per second with only 4 W of power consumption, which is 45 times lower than an NVIDIA GTX 1080Ti.

One solution proffered by developers such as NVIDIA, are embedded GPUs for mobile applications. The Jetson family of compute modules and the NVIDIA TensorRT library provide a power-efficient platform for accelerating deep learning architectures, some of which have already started being used in agriculture [8], [28]–[30]. Although embedded GPUs offer substantial power improvements over conventional GPUs, they are still relatively power hungry when compared to FPGAs. Moreover, they are usually cost prohibitive and are not suitable for developing products that require mass production, such as inference engines on IoT edge devices [31], [32]. Functionally verified HDL implementations developed for FPGA platforms can easily be synthesized to target Application Specific Integrated Circuits (ASICs), yielding considerable reductions in production costs. With recent advancements in HLS techniques, the development of FPGA-accelerated DNNs has been greatly expedited. FPGA-based DNNs present substantial advantages in energy efficiency over

conventional GPU accelerated networks, while exhibiting marginal performance degradation [33]–[39].

## III. PRELIMINARIES

This section briefly reviews the algorithms and methods used in our developed networks for the DeepWeeds classification benchmark.

### A. SOFTMAX REGRESSION & CROSS ENTROPY LOSS

The Softmax model is commonly used to apply logistic regression to multinomial problems. Softmax regression [40] determines a discrete probability distribution of each class, $\rho_i$, for the output of the final layer in our developed deep networks using Equation (1).

$$\rho_i = \frac{e^{y'_i}}{\sum_{i=1}^{N} e^{y'_i}}, \tag{1}$$

where, $y'_i$ represents the predicted class. In addition, $\sum_{i=1}^{N} \rho_i = 1$ where N is the number of classes to be distinguished. Softmax regression is commonly used in tandem with cross-entropy loss, presented in Equation (2), to enable the network to learn different classes during backward propagation where $y_i$ represents the class label.

$$\frac{-1}{N} \sum_{i=1}^{N} y'_i \cdot \log(y_i) + (1 - y'_i) \cdot \log(1 - y_i) \tag{2}$$

### B. BINARY WEIGHT REGULARIZATION

Since the solution space of DNNs is very broad, networks adopting gradient descent optimization algorithms, such as stochastic gradient descent with momentum, are susceptible to overfitting to training data. Overfitting significantly affects the generalization ability of the network and can lead to poor performance on test data. Regularization is any modification made to a learning algorithm to reduce its generalization error, but not its training error.

Binary weight regularization, as proposed in [41], constrains network weights to either $+1$ or $-1$ during forward and backward propagations. The binarization operation transforms the full-precision weights into binary values. Deterministic binarization is based on the sign function presented in Equation (3).

$$w_b = \begin{cases} -1 & \text{if } w \leq 0 \\ +1 & \text{otherwise,} \end{cases} \tag{3}$$

where $w_b$ is the binarized weight and $w$ is the real-valued full-precision weight.

### C. L2 AND BINARYNET REGULARIZATION LOSS TERMS

Regularization is usually added as a term to the learning loss function, to introduce another degree of control to the network parameter growth and help avoid the network over- and under-fitting to the training data.

**Algorithm 1** Training Algorithm of the Accelerated Binarized Neural Networks

**Input:** a mini-batch of (inputs, targets), previous parameters $w_{t-1}$ and $b_{t-1}$, and a learning rate $\eta$.
**Output:** updated parameters $w_t$ and $b_t$.

1: **Forward Propogation**
$w_b \leftarrow$ binarize $(w_{t-1})$.
For $k = 1$ to $L$, compute $a_k$ knowing $a_{k-1}, w_b, b_{t-1}$.
2: **Backward Propogation**
Initialize output layer's activation gradient $\frac{\partial C}{\partial a_L}$
For $k = L$ to 2, compute $\frac{\partial C}{\partial a_{k-1}}$ using $\frac{\partial C}{\partial a_k}$ and $w_b$.
3: **Parameter Update**
Compute $\frac{\partial C}{\partial w_b}$ and $\frac{\partial C}{\partial db_{t-1}}$, using $\frac{\partial C}{\partial a_k}$ and $a_{k-1}$.
$w_t \leftarrow \text{clip}(w_{t-1} - \eta \frac{\partial C}{\partial w_b})$
$b_t \leftarrow b_{t-1} - \eta \frac{\partial C}{\partial b_{t-1}}$.
4: **Weight Normalization**
$w \leftarrow \text{clip}(w)$

Several different regularization techniques have been proposed in literature such as $\ell 1$ and $\ell 2$ regularization [42], dropout [43], and data augmentation [44]. Here, we utilize $\ell 2$ regularization for implementations employing full resolution weights, and BinaryNet-regularization for our implementations employing binarized weights. The $\ell 2$-regularization term is defined in Equation (4).

$$J(\boldsymbol{w}, \boldsymbol{b}) = L(\boldsymbol{w}, \boldsymbol{b}) + D\|\boldsymbol{w}\|_2^2 = L(\boldsymbol{w}, \boldsymbol{b}) + \frac{D}{2}\sqrt{\sum_{i=1}^{N} \boldsymbol{w}_i^2} \tag{4}$$

Here, $J(\boldsymbol{w}, \boldsymbol{b})$ shows the overall loss function that includes a task-related loss term, $L(\boldsymbol{w}, \boldsymbol{b})$, summed with a regularization term, in which $D$ determines the relative significance of the regularization, N denotes the total number of trainable network parameters, $w$ represents a weight, and $b$ is a bias. This regularization technique penalizes the large network weights by adding the sum of their square to the loss function, which helps prevent over-fitting. However, as Equation (4) is differentiable at $w_i = 0$, $\ell 2$-regularization has a non sparse solution, and does not perform feature selection.

BinaryNet-regularization [45] can be defined as Equation (5),

$$J(\boldsymbol{w}, \boldsymbol{b}) = L(\boldsymbol{w}, \boldsymbol{b}) + D\sum_{i=1}^{N}(1 - \boldsymbol{w}_i^2), \tag{5}$$

where all the parameters are similar to Equation (4). Using BinaryNet-regularization, the weights are piloted to +1 or -1, rather than to 0 as in a $\ell 2$-regularization, which is suitable for binary networks.

**TABLE 1:** DeepWeedsX class distribution.

| Class | Species Label | Training | Test | Total |
|---|---|---|---|---|
| 0 | Chinee Apple | 964 | 161 | 1,125 |
| 1 | Lantana | 912 | 152 | 1,064 |
| 2 | Parkinsonia | 884 | 147 | 1,031 |
| 3 | Parthenium | 876 | 146 | 1,022 |
| 4 | Prickly Acacia | 910 | 152 | 1,062 |
| 5 | Rubber Vine | 865 | 144 | 1,009 |
| 6 | Siam Weed | 921 | 153 | 1,074 |
| 7 | Snake Weed | 871 | 145 | 1,016 |
| 8 | Negatives | 7,804 | 1,301 | 9,105 |
| **Total** | | **15,007** | **2,501** | **17,508** |

### D. TRAINING ALGORITHM

Algorithm 1 provides a high-level overview of the training algorithm of our accelerated binarized network architectures. Here, $w$, $b$, and $\eta$ represent the weights, biases, and learning rate, while $C$ denotes the cost function for each mini-batch. Furthermore, $w_b$ represents binary weights and $a_k$ represents the $k$th layer activation function, while binarize() implements Equation (3), and clip() clips values between $-1$ and $+1$.

## IV. DEEPWEEDSX DATASET

The images used to construct the DeepWeedsX dataset have previously been made openly accessible [8], however, they have not been labeled as test and training images. Instead, they have been used in a 5-fold cross validation configuration for training and validation in [8]. Here, we present a labeled variant of DeepWeeds, DeepWeedsX, with clearly defined training and test datasets. We use a splitting ratio of 6:1 (train: test), similar to the popular MNIST [46], CIFAR-10 [47], and CIFAR-100 [47] image classification datasets.

The class distribution of the DeepWeedsX dataset is presented in Table (1). A validation subset may be constructed for parameter optimization using a subset of the labeled training data. In order to facilitate future comparisons to this work, DeepWeedsX, including all its labels and images, is made publicly available. In addition, we have developed data-loaders for PyTorch and TensorFlow, which are intended to assist utilizations in new deep learning experiments[1]. It is worth noting that, one image was removed from the original DeepWeeds dataset [8] to ensure the training and test subsets have the same class distributions.

**TABLE 2:** Color channel mean and standard deviation values used.

| Image Channel | Mean | Standard Deviation |
|---|---|---|
| Red | 0.485 | 0.229 |
| Green | 0.456 | 0.224 |
| Blue | 0.406 | 0.225 |

## V. IMAGE PRE-PROCESSING

Images available from our DeepWeedsX dataset have a resolution of 256×256 pixels. In order to enhance the testing accuracy, several pre-processing steps can be undertaken before the presentation of the images to the network. All of our implementations adopt one of two image pre-processing techniques that are denoted using *Image Pre-processing Techniques (IPT)*, and *Further Image Pre-processing Techniques (FIPT)*.

### A. IMAGE PRE-PROCESSING TECHNIQUES (IPT)

IPT down-samples input images from the native resolution of 256×256 to a resolution of 224×224, 64×64, or 32×32 pixels. No further image pre-processing techniques are performed.

### B. FURTHER IMAGE PRE-PROCESSING TECHNIQUES (FIPT)

FIPT, similarly to [22], randomly crops all images from a resolution of 256×256 to a resolution of 224×224 pixels. This is to ensure that the input images have the same size as the images in well-known datasets such as ImageNet [22], which facilitates using networks developed for those datasets to be deployed for our DeepWeeds images. After all images are cropped, they are down-sampled to a resolution of 64×64, or 32×32 pixels. Their orientation is randomly rotated between -15 and +15 degrees. Their image brightness, contrast, and saturation are also randomly varied by 10%, and normalized between the values of 0 and 1. Finally, the color channels of each image are normalized using distinct mean and standard deviation values as seen in Table (2), which have demonstrated significant performance on the ImageNet dataset.

## VI. NETWORK ARCHITECTURE

The complete structure of the implemented networks consists of two main components: the software and the hardware. The software architecture defines the targeted neural network structures, which are described in C++ and OpenCL kernels. The hardware architecture describes the integration of the hardware used to run the OpenCL kernels, and a host controller, which is the program executed on a host processor to launch OpenCL kernels and to manage available memory.

### A. SOFTWARE ARCHITECTURE

Three popular deep network architectures, i.e. VGG-16 [48], DenseNet-128-10 [49], with 128 layers and a growth rate of 10, and Wide Residual Network (WRN)-28-10 [50], with 28 layers and a growth rate of 10, were trained using full resolution and deterministically binarized weights for comparison. Although networks with tuned hyper-parameters would be expected to achieve higher validation accuracies, we present baseline implementations on our labeled dataset without any hyper parameter tuning. The mentioned networks are chosen as they demonstrate

significant performance on the ImageNet dataset, which we believe to be indicative of high performance on the DeepWeedsX dataset. In favour of reproducible research, we have made the specific code level implementations of all these networks publicly available through a Github repository[1].

More details on our developed software network architectures are as follows. The output of each network's last layer is fed through a Softmax activation layer [40], and each network's loss is minimized using cross-entropy. Stochastic gradient descent with momentum [51] is used to optimize network parameters. For all networks, the momentum variable $m$, is set to 0.8. In our implementations, BNNs employ a smaller initial learning rate, $\eta[0] = 0.001$, to avoid frequent sign changes, while for conventional networks using full resolution weights a larger initial learning rate of $\eta[0] = 0.01$ is used to speed-up learning convergence. In addition, the regularization coefficient, $D = 5e^{-7}$ for our BinaryNet-regularization terms and $D = 1e^{-5}$ for our $\ell 2$-regularization terms. For all BNNs, ReLU activation functions, used in conventional networks, are replaced with the hyperbolic tangent function, $tanh$.

Furthermore, in order to maximize the networks' performance, a decaying learning rate is used during training for all networks. This learning rate, $\eta$, is decayed by a factor of ten when learning falls stagnant, i.e. does not increase for a period of 10 epochs. Finally, the weights are randomly uniformly distributed using the He initialization technique presented in [52] to accelerate learning convergence for full resolution weights used for the parameter update stage.

### B. HARDWARE ARCHITECTURE

After functionally verifying our implementations using the *PyTorch* [53] ML library with a state-of-the-art *Titan V* GPU and an *AMD Ryzen 2700X @ 4.10 GHz Overclocked (OC)* CPU, we developed hardware architectures consisting of C++ host controllers and multiple OpenCL kernels, which were accelerated using FPGAs and GPUs. For x86-based systems, OpenCL accelerated kernels using FPGAs typically reside on an FPGA development board, which is connected to a separate independent host system through the PCIe express interface [35]. For ARM-based systems, the FPGA is typically connected to a Hard Processor System (HPS) on a SoC through specialized bridges – as in the case of the Intel DE1-SoC development board used herein. This allowed our proposed FPGA-accelerated networks to run completely independently using the SoC, without using a separate device for computation.

Our OpenCL implementations originate from the publicly available *DeepCL* OpenCL ML library[2]. All convolutional, inner-product, activation, pooling, regularization, and batch-normalization operations are described using single work-item kernels with an NDRange size of (1, 1, 1). Multi-mode 3D NDRange kernels are used to fetch and store data to and from the global memory for all computation pipelines, similarly to [35]. Consequently, our implementations operate with minimal controller computation. We make these efforts toward an eventual implementation that avoids controller computation completely, with motivations to make our future designs applicable to both GPUs and non-SoC FPGAs, avoiding the power overhead of the host controller, that is required for the OpenCL programming model. In the following subsections, we detail the different synthesis constructs, such as the loop unroll factor (#pragma unroll for GPU), and Single-Instruction-Multiple-Data (SIMD) vectorization factor, used for our developed single work-item kernels targeted for acceleration on heterogeneous platforms.

#### 1) Convolutional Kernel

Convolutional operations were implemented by mapping 3-D convolutions as matrix multiplication operations, by flattening and rearranging the input features, similarly to [54]. Each work-item performs either fused multiply and accumulate (MAC) operations, or accumulate operations, on the local memory data, depending on whether weights are quantized to binary states or not. This process is accomplished using the loop unrolling technique, and is repeated by sliding the convolutional window to get the corresponding elements in the product matrix. We further detail the XNOR kernels, described using Register Transfer Level (RTL) modules, utilized in our FPGA BNN implementations for convolutional and FC kernels in Section VI-C.

#### 2) Inner-product Kernel

Inner-product operations were implemented for Fully Connected (FC) layers using single work-item kernels, where, similarly to our convolutional kernel, fused MAC, or accumulate operations are performed, depending on whether weights are quantized to binary states or not.

#### 3) Activation Kernel

All activation functions were computed at the output of convolution and inner product implementations using single work-item kernels.

#### 4) Pooling, Regularization, & Batch-normalization Kernels

Pooling, regularization, optimization, and batch normalization operations were implemented using single work-item kernels, where acceleration is achieved by unrolling the loop to generate parallel outputs in a single cycle.

### C. PLATFORM SPECIFIC COMPILATION & PERFORMANCE ENHANCING TECHNIQUES

In this section we detail the platform specific compilation details, libraries, and resources required for compilation and

[1] *https://github.com/coreylammie/Low-Power-and-High-Speed-Deep-FPGA-Inference-Engines-for-Weed-Classification*

[2] *https://github.com/hughperkins/DeepCL*

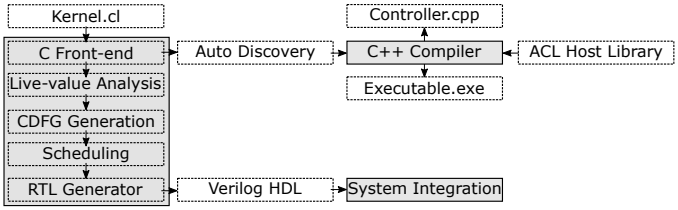


**FIGURE 2:** Compilation flow for the IOC for our OpenCL implementations on FPGA.

deployment of our developed host controller and OpenCL kernels, for FPGA and GPU platforms.

Both our GPU and FPGA implementations of BNNs are accelerated using XNOR kernels, which enables Single Instruction Multiple Data (SIMD) within a register (SWAR) [41]. Here, full precision 32-bit floating point weights are concatenated into groups of 32 binary variables into 32-bit registers, resulting in a 32-time speed-up on bitwise operations.

### 1) FPGA

To compile OpenCL kernels for FPGA, the *Intel FPGA SDK for OpenCL Offline Compiler* (IOC) was used, as part of the *Intel FPGA SDK for OpenCL* and *Quartus Prime Design Suite* 18.1. IOC fully supports version 1.0 of the OpenCL specification, and has some preliminary support for newer features from version 2.0. Figure (2) presents the compilation flow for the IOC. Here, inputs are a set of OpenCL kernels, and the output is a singular $.aocx$ image file containing the necessary information to program the FPGA at runtime containing the FPGA image. The host application loads data that is used to create program objects, to program the target FPGA, as required for all kernel launch operations.

SWAR was also implemented using a digital logic approach, similarly to [55], to accelerate our FPGA BNN implementations. Figure (3) illustrates SIMD using 32-bit registers. Using RTL modules, the given size of each register to store binary weights is arbitrary and not constrained by the computer architecture. These instructions, therefore, take only four clock cycles on FPGAs.

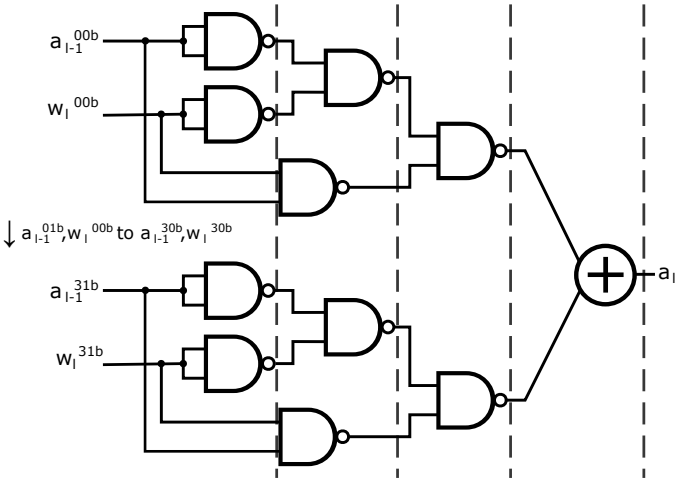


**FIGURE 3:** Schematic of the XNOR kernel implementation on FPGA with 32-bit registers containing 32 binary variables. Here, vertical lines represent clock cycles. $a_{l-1}$ contains binary activations, and $w_l$ contains binary weights.

We integrate our developed XNOR RTL module into the Intel FPGA SDK for OpenCL Pipeline using the IOC. Our RTL module has a balanced latency, where its threads match the number of pipeline stages in our design. This allows the threads of the RTL module to execute without stalling the SDK's pipeline and bottlenecking operational performance.

### 2) GPU

In addition to accelerating the targeted DeepWeedsX recognition networks on the Intel DE1-SoC FPGA development board, the networks were also accelerated on a state-of-the-art *Titan V* GPU to execute OpenCL kernels and an *AMD Ryzen 2700X @ 4.10 GHz Overclocked (OC)* CPU to drive the host controller. We use version 419.35 of the Titan V GPU driver to launch compute kernels. Using SWAR OpenCL kernels, on GPUs, it is possible to evaluate 32 network connections with only 3 instructions (accumulation, popcount, and xnor), as described in Equation (6).

$$a_l += \text{popcount}(\text{xnor}(a_{l-1}^{32b}, w_l^{32b})). \tag{6}$$

Here, $l$ denotes the layer number, $a_l$ is the resulting weighted sum, and $a_{l-1}^{32b}, w_l^{32b}$ are the concatenated inputs and weights. These instructions take 6 overall clock cycles, including 1 for accumulation, 4 for popcount, and 1 for xnor, on the *NVIDIA Titan V* GPU used [56].

While NVIDIA's CUDA compiler is, presently, much more efficient and mature than their OpenCL compiler, to advocate fair comparison, we use OpenCL implementations across FPGA and GPU platforms. We note that enhanced timing performance is expected on NVIDIA GPUs using CUDA alongside with NVIDIA's Deep Neural Network library (cuDNN).

| Network Architecture | (3, 32, 32) | (3, 64, 64) | (3, 224, 224) |
|---|---|---|---|
| | IPT | | |
| VGG-16 | **86.72** | 89.48 | 91.08 |
| DenseNet-128-32 | **90.08** | 91.52 | 89.40 |
| WRN-28-10 | **88.88** | 93.36 | 94.82 |
| | FIPT | | |
| VGG-16 | **81.45** | 89.12 | 93.04 |
| DenseNet-128-32 | **85.89** | 86.05 | 94.24 |
| WRN-28-10 | **85.97** | 90.72 | 95.85 |

**TABLE 3:** Maximum validation accuracy achieved during 200 epochs of training for all baseline implementations used to investigate down-sampling performance degradation.

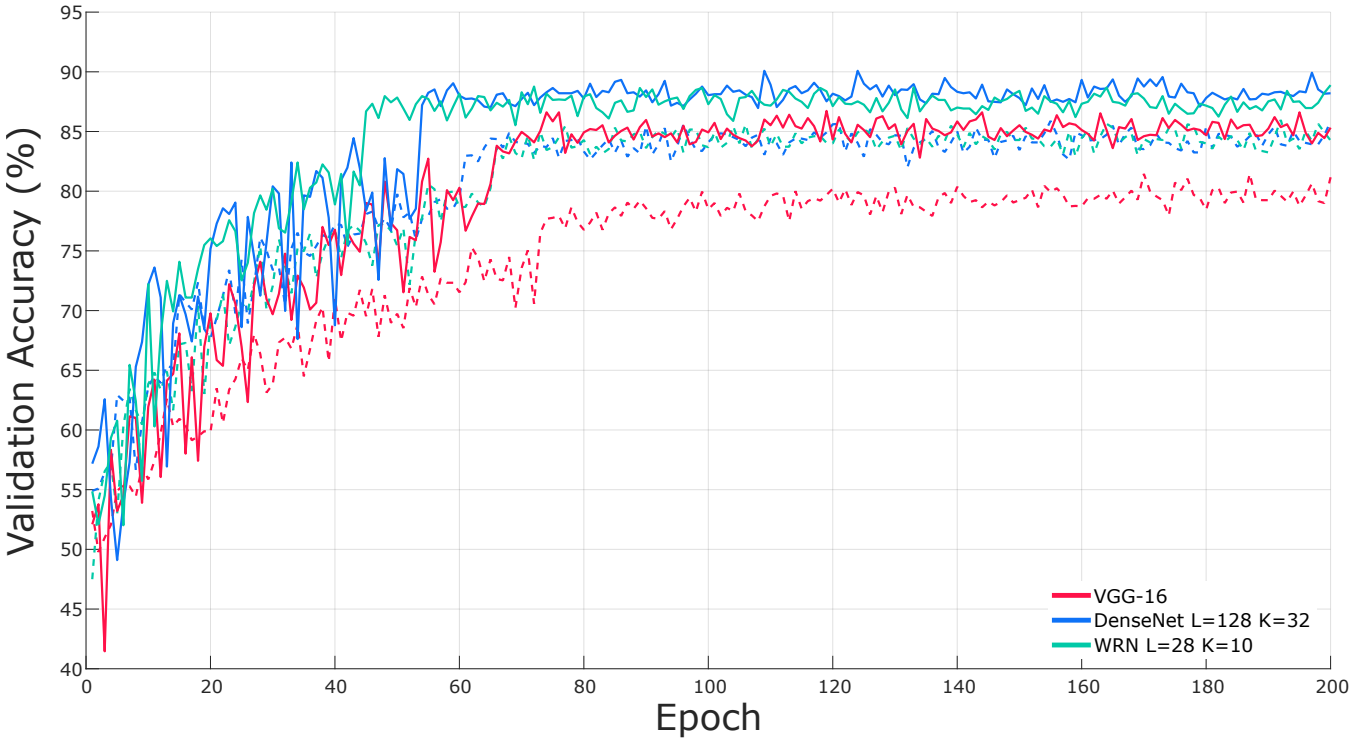


**FIGURE 4:** Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 32, 32). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

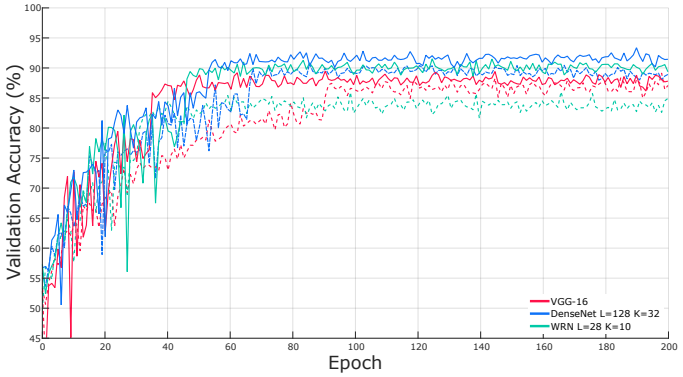


**FIGURE 5:** Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 64, 64). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

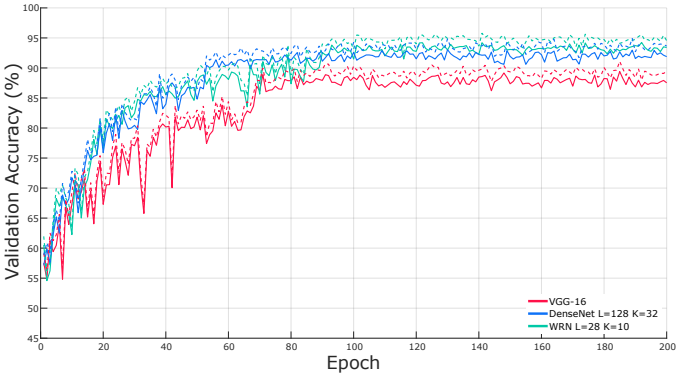


**FIGURE 6:** Validation accuracy of all baseline implementations used to investigate down-sampling and image pre-processing effect on performance during 200 training epochs with inputs down-sampled to (3, 224, 224). Solid lines represent implementations adopting IPT. Dashed lines represent implementations adopting FIPT.

## VII. INVESTIGATING THE EFFECT OF IMAGE DOWN-SAMPLING AND PREPROCESSING ON PERFORMANCE

Before gathering implementation results, VGG-16 [48], DenseNet-128-32 [49], and WRN-28-10 [50] were trained using full resolution weights with a batch size, $\Im = 32$, for input images down-sampled to three different sizes. We restrict the batch size to 32 and below due to the harsh real-time constraints presented by our specific edge computing use case for robotic weed control, which is discussed further in Section IX-B. These sizes include dimensions of $32\times32$, $64\times64$, and the original size of $224\times224$. This was performed to investigate the down-sampling effect on the degradation of our chosen deep networks validation accuracy. Figures (4), (5), and (6) demonstrate the validation accuracy of all networks over 200 training epochs. While other works report results over a 500 epoch training routine, we observed that learning on the training set was saturated at around 150 epochs and therefore report results over 200 epochs. The maximum

| Network Architecture | (3, 32, 32) | (3, 64, 64) | (3, 224, 224) |
|---|---|---|---|
| VGG-16 | 33,642,569 | 39,934,025 | 49,225,881 |
| DenseNet-128-32 | 7,727,065 | 7,757,737 | 8,217,817 |
| WRN-28-10 | 36,478,553 | 36,478,553 | 36,478,553 |

**TABLE 4:** Total number of trainable network parameters for each network architecture.

validation accuracy achieved for each implementation is presented in Table (3). It is worth noting that, for the networks shown in Table (3), no hyper-parameter optimization is performed. In addition, for the FIPT cases, we replicated our image pre-processing steps originally proposed in our previous work [8], which demonstrated significant performance for input images with a resolution of 224×224. It is expected that, hyper-parameter optimization, different types or amounts of image pre-processing techniques, or a combination of them could lead to validation accuracy improvement.

From Figures (4), (5), and (6) it can be observed that down-sampling input images to (3, 32, 32) leads to performance degradation, compared to using images with (3, 64, 64) or those with a native resolution of (3, 224, 224). In addition, the figures show that for low-resolution images, i.e. (3, 32, 32) and (3, 64, 64), performing FIPT leads to lowering accuracy. FIPT is useful in the case of the native resolution images and leads to the highest accuracy achievable.

Interestingly, it is possible to achieve >85% validation accuracy using down-sampled input images with IPT at (3, 32, 32), which can significantly improve processing speed and reduce the total required memory utilization during inference, compared to the use of higher resolution images. Therefore, we down-sample all images to (3, 32, 32) to obtain all our implementation results reported in the following sections. This is expected to drastically reduce the training time required, as well as the resource utilization for all networks, which have previously been demonstrated to be governed by the total number of trainable network parameters, input image size, and network configuration [57].

Table (4) presents the total trainable network parameters for each network architecture used to obtain our implementation results. For VGG-16 and DenseNet-128-32 architectures, additional parameters are required for larger resolution input images to implement additional max pooling and fully connected layers. However, the WRN-28-10 network parameter size shows no dependency to the input image resolution, which could be attributed to its wide (not deep) structure.

## VIII. IMPLEMENTATION RESULTS

Initially, conventional networks trained using full resolution weights were all implemented on the GPU platform (see Table 3). This was to have as a baseline for validation accuracy comparison to both previous works and our binarized implementations. Direct comparison to the relevant previous work [8] is not possible using the given labeled dataset (DeepWeedsX), because, in [8] five-fold cross-validation is used to report accuracy, while here we obtained accuracy using a train-test dataset split. Nonetheless, the best validation accuracy achieved here (95.85%) using WRN-28-10 with FIPT, is marginally better than the best accuracy previously achieved using ResNet-50 with FIPT (95.7%) as reported in [8].

In order to validate and investigate the performance of the proposed FPGA- and GPU-accelerated BNN architectures on the DeepWeedsX dataset, the validation error rate, power consumption, and Inference Time Per Image (ITPI) were analyzed. On both FPGA and GPU platforms, all aforementioned performance metrics were determined during inference, i.e. after importing trained weights, for different batch sizes $\Im \in [4, 8, 16, 32]$ to investigate the effect of the batch size on inference performance. As discussed earlier, we restrict the batch size to between 4 and 32 due to the harsh real-time constraints presented by our specific edge computing use case for robotic weed control (see Section IX-B). For FPGA implementations we also report the resource utilization, which is an important parameter in identifying hardware cost.

The total kernel power usages were determined using the *Post Place & Route Estimator* for FPGA post-synthesis, and *NVIDIA-SMI* for GPU. To ensure all reported power usage readings are accurate for our GPU implementations, we artificially elongate kernel execution times when measuring GPU kernel power utilization.

It is worth noting that, there is no need to consider the time required for image pre-processing because for all (3, 32, 32) cases, networks employing IPT outperform their FIPT counterparts (see Table 3). It is expected that during real-time inference, images are fed directly to the inference engine after undergoing pipelined real-time image resizing similar to [58], hence, requiring no down-sampling (IPT). There is also no need for FIPT, because FIPT is only useful for (3, 224, 224). We used images of size (3, 32, 32).

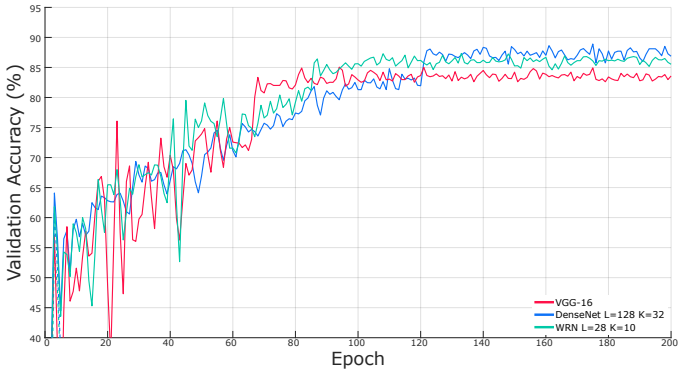


**FIGURE 7:** Validation accuracy after each epoch during 200 training epochs for our new binarized implementations with down-sampled (3, 32, 32) images.

| Baseline Network Architecture | Total Kernel Power Usage (W) | Inference Time per Image (ms) | Validation Accuracy (%) |
|---|---|---|---|
| | $\Im = 32$ | | |
| VGG-16 | 41.00 | 5.038 | 86.72 |
| DenseNet-128-32 | 41.00 | 4.398 | 90.08 |
| WRN-28-10 | 44.67 | 2.535 | 88.88 |

**TABLE 5:** Implementation results obtained from our baseline implementations using full resolution weights, trained over 200 epochs with down-sampled (3, 32, 32) images. During inference $\Im = 32$.

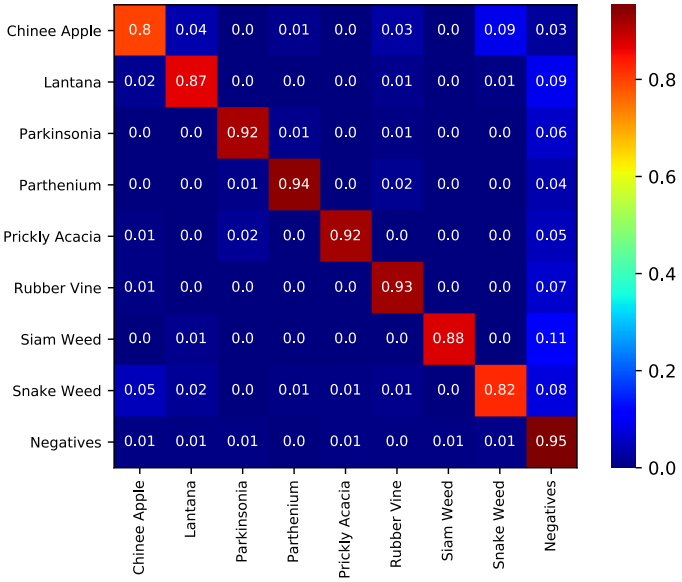


**FIGURE 8:** Confusion matrix obtained using the DenseNet L=128, K=32 with baseline implementation. Performance is described for each individual class. Values are normalized from 0 to 1 corresponding to 0% and 100% validation accuracy, respectively.

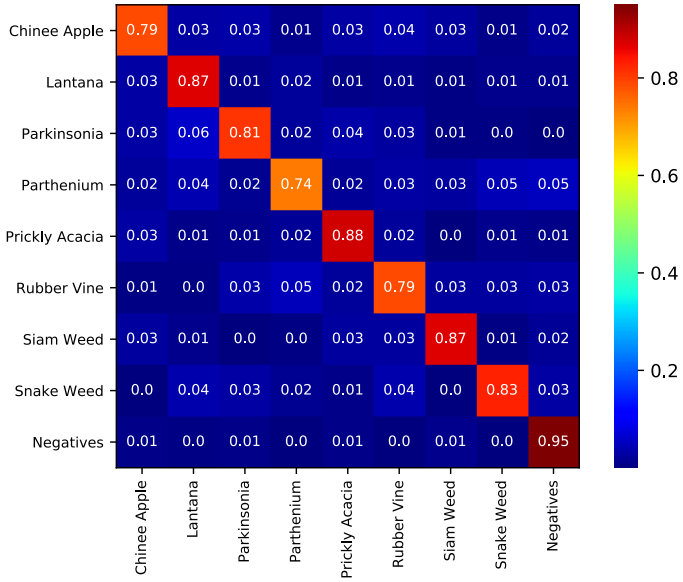


**FIGURE 9:** Confusion matrix obtained using the binary DenseNet L=128, K=32 with IPT. Performance is described for each individual class. Values are normalized from 0 to 1 corresponding to 0% and 100% validation accuracy, respectively.

### A. BASELINE IMPLEMENTATIONS

Baseline implementations of VGG-16 [48], DenseNet-128-32 [49], and WRN-28-10 [50], were trained using full resolution weights with $\Im = 32$, on GPU for images of size (3, 32, 32). The collected performance metrics are reported in Table 5. It can be observed that DenseNet L=128, K=32 achieves the highest validation accuracy of 90.08%. It requires 0.64 ms less inference time per image, compared to its VGG-16 variant, while consuming the same amount of power and yielding an increase 0f 3.36% in validation accuracy. Compared to WRN-28-10 network, the DenseNet L=128, K=32 achieves 1.2% improvement in accuracy, while being 1.86 ms slower in image inference, but consuming 3.67 W less power.

Furthermore, in order to determine class specific performance, a confusion matrix for the top performing baseline implementation, DenseNet L=128, K=32, is presented in Figure (8). It can be observed that the individual class accuracy is weakly correlated to the class distribution in the DeepWeeds dataset. Further species-specific performance discussion is presented in Section IX-A.

### B. BINARY IMPLEMENTATIONS

We implement and train our new binary networks of the selected deep architecture across GPU and FPGA platforms. The validation accuracy for each epoch during training for each implementation is presented in Figure (7). As all networks are trained using the same *NVIDIA Titan V* GPU, only one plot for each network is presented. Consequently, the validation accuracies and corresponding confusion matrices are identical across platforms.

From Figure (7), it can be observed that, similar to our baseline implementations adopting full resolution weights, binary DenseNet L=128, K=32 achieves the highest validation accuracy, with a degradation of only 1.17% compared to its full-resolution counterpart.

In addition, Table (6) reports the total kernel power usages and inference times for both GPU and FPGA implementations for various batch sizes of $\Im \in [4, 8, 16, 32]$. Table (6) demonstrates that as the batch size, $\Im$, is increased, the inference time per image is notably decreased. We believe this is a a direct result of increased parallelism.

For all networks, the FPGA implementations require less time to perform inference despite operating at a much lower

| Binary Implementation | | VGG-16 | | | | DenseNet-128-32 | | | | WRN-28-10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ |
| Total Kernel Power Usage (W) | GPU | 46.16 | 47.10 | 47.39 | 47.69 | 56.78 | 56.83 | 57.47 | 58.71 | 55.64 | 56.68 | 56.72 | 58.07 |
| | FPGA | 4.19 | 4.58 | 5.31 | 6.26 | 4.48 | 4.72 | 5.40 | 6.72 | 4.01 | 4.49 | 5.26 | 6.04 |
| Inference Time per Image (ms) | GPU | 9.87 | 5.55 | 4.5 | 3.3 | 7.47 | 3.93 | 2.88 | 2.43 | 4.83 | 2.98 | 2.38 | 1.49 |
| | FPGA | 6.647 | 3.843 | 2.974 | 2.199 | 4.762 | 2.689 | 2.045 | 1.539 | 3.268 | 2.02 | 1.539 | 1.059 |
| Validation Accuracy (%) | | 85.05 | | | | 88.91 | | | | 87.33 | | | |

**TABLE 6:** Implementation results obtained from our new binarized implementations with down-sampled (3, 32, 32) images, over 200 epochs.

| Binary Implementation | VGG-16 | | | | DenseNet-128-32 | | | | WRN-28-10 | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ | $\Im = 4$ | $\Im = 8$ | $\Im = 16$ | $\Im = 32$ |
| Flip Flops | 33.34% | 48.49% | 68.72% | 96.17% | 28.09% | 36.82% | 71.49% | 74.67% | 30.17% | 43.07% | 58.63% | 88.96% |
| ALMs | 58.32% | 61.91% | 87.46% | 96.30% | 35.22% | 46.94% | 78.49% | 92.92% | 50.26% | 53.29% | 79.74% | 88.89% |
| DSPs | 74.11% | 80.14% | 85.42% | 95.20% | 34.73% | 47.26% | 53.53% | 68.81% | 55.92% | 75.29% | 84.13% | 90.35% |
| Frequency [MHz][1] | 52 | 46 | 43 | 39 | 58 | 49 | 43 | 41 | 51 | 45 | 44 | 41 |

**TABLE 7:** Device utilization (%) comparison for the implemented FPGA-accelerated BNNs adopting IPT. All reported hardware utilization numbers are expressed as percentages of the total available resources on the FPGA. [1]The maximum frequency for each implementation was extracted from *acl_quartus_report.txt* report, generated by the Quartus Design Studio.

operational frequency. Our DenseNet L=128, K=32 implementation on FPGA reduces the inference time per image compared to its GPU counterpart by 0.89 ms for $\Im = 32$ while achieving the same validation accuracy, and by 2.86 ms compared to its baseline implementation time reported in Table (5). These results and their implications are discussed in Section IX-B.

Furthermore, in order to determine class specific performance of binarized networks on GPU and FPGA, a confusion matrix for the top performing implementation, DenseNet L=128, K=32 is presented in Figure (9). Interestingly, the class specific performance for binarized DenseNet varies significantly across various weed species, while degrading only 1.17% overall. Section IX-A provides further discussion on species-specific performance.

In addition, to investigate the hardware complexity of the implemented binarized networks on FPGA and compare it to the achieved inference and power consumption figures, the device utilization of all FPGA accelerated networks were measured and presented in Table (7).

From Table (7) it can be observed that the device utilization for each network architecture is weakly correlated with the batch size during inference. For all our implementations, as $\Im$ is increased, the Flip Flops, ALMs, DSPs required for synthesis are increased and the maximum synthesizable frequency is decreased. Our best performing implementation, DenseNet L=128, with $\Im = 32$, requires 46.58% more Flip Flops, 57.7% more ALMs, and 34.08% more DSPs than its $\Im = 4$ counterpart.

## IX. FURTHER DISCUSSION

Here we provide further discussion of our implementation results and how they benefit the application of robotic weed control.

### A. CLASSIFICATION PERFORMANCE

Tables (5) and (6) show that our baseline and binary implementations of the DenseNet architecture consistently outperform VGG-16 and WRN-28-10. With images drastically down-sampled to (3, 32, 32), our full precision baseline DenseNet-128-32 implementation offers a validation accuracy of 90.1% on the DeepWeedsX dataset. This compares well with the ResNet-50 architecture used in [8] to achieve 95.7% performance on much larger images, $224 \times 224$ pixels in size.

However, validation accuracy is an unreliable sole metric due to the imbalanced nature of the DeepWeeds and DeepWeedsX datasets. In the application of robotic weed control, the goal is to maximize coverage of weed targets at the cost of collateral off-target damage. As such, it is vitally important to consider the performance on each individual species. The confusion matrix of the baseline implementation of DenseNet, presented in Figure (8), is analogous to the classification performance in [8]. The species with the highest recall accuracy, ranging from 87-95%, include: Lantana, Parkinsonia, Parthenium, Prickly acacia, Rubber vine, Siam weed and negative plant life. The species presenting the most difficult challenge for the network, ranging from 80-82% recall accuracy, are Chinee apple and Snake weed. With the model confusing 9% of Chinee apple as Snake weed, and 5% vice versa, we reason their respective poor performance is due to the inherent similarity in the two plants image features. The similarity between these two classes is made evident in Figure (10), which presents an example image of each class.

Fortunately, confusing one weed target for another is inconsequential in the application of robotic weed control. However, missed targets (i.e. false negatives) and off-target damage (i.e. false positives) are of great consequence so we must examine the performance on the negative class. The baseline DenseNet architecture confuses between 3-11% of each species with the negative class. This constitutes a


**FIGURE 10:** Example images from each class of the *DeepWeeds* dataset, including: (a) Chinee apple, (b) Lantana, (c) Parkinsonia, (d) Parthenium, (e) Prickly acacia, (f) Rubber vine, (g) Siam weed, (h) Snake weed and (i) Negatives. Chinee apple and Snake weed are prone to high levels of confusion due to their very similar features.

significant number of missed targets. Also, 5% of the negative class is falsely classified as positive species. The existence of false positives is assumed to be the result of the massive variation of plant life in the negative class.

Table (6) reveals that the best performing binarized implementation is again the DenseNet-128-32 architecture, which offers 88.9% average classification accuracy. Interestingly, the inter-species performance is vastly different to the baseline implementation, as shown by the confusion matrix in Figure (9). Our binarized implementation of the DenseNet architecture appears to have generalized the classification performance of the network across species. Confusions between species are more scattered and are no longer owing to visibly discernible characteristics. The species offering the best performance, ranging from 83-88%, include: Lantana, Prickly acacia, Siam weed and Snake weed. While the species presenting the most difficult challenge, ranging from 74-81% are: Chinee apple, Parkinsonia, Parthenium and Rubber vine. We believe that the extreme quantization of weights to binary states prevented less-distinctive features to be extracted, causing a degradation in validation accuracy for weed species with a large number of features. Species with distinctive features, or lack thereof, demonstrated similar accuracies to our baseline implementations. For example, in both our best performing implementations the negative class is strongly classified at 95% with 5% false positives.

Tables (5) and (6) also show that the performance of all binarized implementations are slightly worse than their full precision baseline counterparts. This result confirms what is seen in literature. However, the real-time performance of these binarized networks are far greater than their full precision counterparts and present a fruitful tradeoff for the application of robotic weed control, as discussed below.

### B. REAL TIME PERFORMANCE

Tables (5) and (6) show that every novel binarized implementation presented here significantly outperforms its GPU baseline implementation in terms of inference time and

required power, with only a slight degradation in validation accuracy. The WRN architecture offers the fastest inference engine with a 1.018 ms inference time when implemented on an FPGA. While DenseNet-128-32, our most accurate architecture, also offers a significant increase in speed, with an average inference time of 1.539 ms per image.

In addition to the binarization of full precision architecture and acceleration using FPGA hardware, two further methods of pushing the real-time performance were investigated: presenting a smaller image size to the network, and reducing the amount of pre-processing or image augmentations performed. Table (3) reveals that down-sampling the image size from (3, 224, 224) to (3, 32, 32) only slightly decreases the average validation accuracy from 94.24% to 90.1% for DenseNet, 93.04% to 86.7% for VGG-16 and 95.85% to 88.9% for WRN. The significantly smaller image size of (3, 32, 32) is a major reason these networks perform inference faster, compared to the (3, 224, 224) shaped architectures in [8].

Table (3) also shows that applying further image pre-processing techniques offers no advantage in classification performance for the utilized down-sampled images. In fact, we observed that validation accuracy is degraded by an average of over 4% for the binary implementations when images are down-sampled below (3, 224, 224) when further image augmentations are applied. This suggests that our FIPT are ill-suited to low resolution images and can only be beneficial to large-resolution images, which we do not use here. However, as discussed in Section VII, these results are expected to be significantly improved if further image preprocessing technique investigation, and/or hyper-parameter optimization is performed, for each network, and each input image size.

Let us consider the use case of the prototype agricultural robotic spot-sprayer, AutoWeed, first introduced in [8]. Its optical system comprises four FLIR Blackfly 23S6C high-resolution cameras, each providing a 450 x 280 mm field of view with a maximum data rate of 41 fps. This allows a threshold of at most 100 ms total processing time per image per camera for the selective spot sprayer to operate at the target speed of 10 km/hr. This imposes a required frame rate of at least 10 fps per camera to achieve target real-time performance. With simultaneous image acquisition from four cameras, batch sizes of between 4 and 32 were considered for our above implementations. The lower limit of 4, considers processing one frame at a time from each camera. While the upper limit of 32, considers waiting for the acquisition of up to 8 frames from each camera before processing them in a batch for the apparent per image inference speed increase.

With an average inference time of 1.539 ms implemented on an FPGA, the $450 \times 280$ mm field of view of our specific edge device can be processed fast enough to achieve a frame rate of over 600 fps. Far exceeding the real time requirement for one camera at 10 km/hr. This efficiency would also allow the robot to operate at a much higher vehicle speed to yield more efficient performance in the agricultural domain.

Compared to existing full precision architectures and their power-hungry GPU implementation [8], the low-power and high-speed inference engines presented here offer an attractive tradeoff with slightly worse classification performance for greatly increased speed and power efficiency. This tradeoff will allow researchers and developers to solve the speed and power inefficiencies in applications of precision agriculture, like robotic weed control, with software instead of hardware. The proposed binarized solutions can also be generalized for improving the efficiency of edge computing in general, where slight amounts of accuracy can be traded off for great amounts of speed and power improvement.

## X. CONCLUSION

In this paper, we are the first to investigate the acceleration of binarized DNNs on GPUs and FPGAs using the high-level OpenCL framework for weed species classification targeted toward edge computing applications and robotic weed control. We investigated the performance degradation exhibited when down-sampling input images, and demonstrated that significantly reducing the image resolution has a marginal effect on validation accuracy. After thoroughly comparing efficient implementations on GPU and FPGA platforms, we were able to achieve a $>$7-fold decrease in power consumption, while performing inference on weed images 2.86 times faster while degrading validation accuracy by only 1.17% on our newly labeled and publicly available dataset. Finally, we provided further discussion pertaining to species-specific classification performance and real time performance implications for robotic weed control. The implemented networks demonstrated here represent ideal candidates for future implementations in edge computing devices and precision agricultural robots.

## ACKNOWLEDGEMENTS

We gratefully acknowledge the research funding from the Australian Government Department of Agriculture and Water Resources Control Tools and Technologies for Established Pest Animals and Weeds Programme (Grant No. 4-53KULEI). We also acknowledge the NVIDIA GPU Grant Program for the donation of a *NVIDIA Titan V GPU*. Corey Lammie acknowledges the JCU Domestic Prestige Research Training Program Scholarship (DRTPS). The authors also greatly appreciate the very constructive feedback of the second anonymous reviewer.

## REFERENCES


[1] P. Gonzalez-de Santos, A. Ribeiro, C. Fernandez-Quintanilla, F. Lopez-Granados, M. Brandstoetter, S. Tomic, S. Pedrazzi, A. Peruzzi, G. Pajares, G. Kaplanis, M. Perez-Ruiz, C. Valero, J. del Cerro, M. Vieri, G. Rabatel, and B. Debilde, "Fleets of robots for environmentally-safe pest control in agriculture," Precision Agriculture, vol. 18, no. 4, pp. 574–614, Aug 2017. [Online]. Available: https://doi.org/10.1007/s11119-016-9476-3

[2] C. Fernández-Quintanilla, J. M. Peña, D. Andújar, J. Dorado, A. Ribeiro, and F. López-Granados, "Is the current state of the art of weed monitoring suitable for site-specific weed management in arable crops?" Weed Research, vol. 58, no. 4, pp. 259–272, May 2018. [Online]. Available: https://doi.org/10.1111/wre.12307

[3] D. L. Shaner and H. J. Beckie, "The future for weed control and technology," Pest Management Science, vol. 70, pp. 1329–1339, 2014. [Online]. Available: https://doi.org/10.1002/ps.3706

[4] D. C. Slaughter, D. K. Giles, and D. Downey, "Autonomous robotic weed control systems: A review," Computers and Electronics in Agriculture, vol. 61, pp. 63–78, 2008. [Online]. Available: https://doi.org/10.1016/j.compag.2007.05.008

[5] A. dos Santos Ferreira, D. M. Freitas, G. G. da Silva, H. Pistori, and M. T. Folhes, "Weed detection in soybean crops using ConvNets," Computers and Electronics in Agriculture, vol. 143, pp. 314 – 324, 2017. [Online]. Available: https://doi.org/10.1016/j.compag.2017.10.027

[6] S. H. Lee, C. S. Chan, S. J. Mayo, and P. Remagnino, "How deep learning extracts and learns leaf features for plant classification," Pattern Recognition, vol. 71, pp. 1–13, November 2017. [Online]. Available: https://doi.org/10.1016/j.patcog.2017.05.015

[7] S. H. Lee, C. S. Chan, P. Wilkin, and P. Remagnino, "Deep-plant: Plant identification with convolutional neural networks," in Proceedings of the 2015 IEEE International Conference on Image Processing (ICIP), Québec City, Canada, 2015, pp. 452–456. [Online]. Available: https://doi.org/10.1109/ICIP.2015.7350839

[8] A. Olsen, D. A. Konovalov, B. Philippa, P. Ridd, J. C. Wood, J. Johns, W. Banks, B. Girgenti, O. Kenny, J. Whinney, B. Calvert, M. R. Azghadi, and R. D. White, "Deepweeds: A multiclass weed species image dataset for deep learning," Scientific Reports, vol. 9, no. 1, p. 2058, Feb. 2019. [Online]. Available: https://doi.org/10.1038/s41598-018-38343-3

[9] A. Bakhshipour and A. Jafari, "Evaluation of support vector machine and artificial neural networks in weed detection using shape features," Computers and Electronics in Agriculture, vol. 145, pp. 153 – 160, 2018. [Online]. Available: https://doi.org/10.1016/j.compag.2017.12.032

[10] M. Dyrmann, R. N. Jørgensen, and H. S. Midtiby, "RoboWeedSupport - Detection of weed locations in leaf occluded cereal crops using a fully convolutional neural network," Advances in Animal Biosciences, vol. 8, no. 2, p. 842–847, 2017. [Online]. Available: https://doi.org/10.1017/S2040470017000206

[11] S. G. Wu, F. S. Bao, E. Y. Xu, Y. Wang, Y. Chang, and Q. Xiang, "A leaf recognition algorithm for plant classification using probabilistic neural network," in Proceedings of the 2007 IEEE International Symposium on Signal Processing and Information Technology (ISSPIT), Cairo, Egypt, Dec 2007, pp. 11–16. [Online]. Available: https://doi.org/10.1109/ISSPIT.2007.4458016

[12] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. B. Soares, "Leafsnap: A computer vision system for automatic plant species identification," in Proceedings of the 2012 European Conference on Computer Vision (ECCV), Berlin, Heidelberg, 2012, pp. 502–516. [Online]. Available: https://doi.org/10.1007/978-3-642-33709-3_36

[13] D. Hall, C. McCool, F. Dayoub, N. Sunderhauf, and B. Upcroft, "Evaluation of features for leaf classification in challenging conditions," in Proceedings of the 2015 IEEE Winter Conference on Applications of Computer Vision (WACV), Hawaii, USA, 2015, pp. 797–804. [Online]. Available: https://doi.org/10.1109/WACV.2015.111

[14] C. Kalyoncu and Önsen Toygar, "Geometric leaf classification," Computer Vision and Image Understanding, vol. 133, pp. 102 – 109, 2015. [Online]. Available: https://doi.org/10.1016/j.cviu.2014.11.001

[15] J. Carranza-Rojas, H. Goeau, P. Bonnet, E. Mata-Montero, and A. Joly, "Going deeper in the automated identification of herbarium specimens," BMC Evolutionary Biology, vol. 17, p. 181, 2017. [Online]. Available: https://doi.org/10.1186/s12862-017-1014-z

[16] A. Shirzadifar, S. Bajwa, S. A. Mireei, K. Howatt, and J. Nowatzki, "Weed species discrimination based on SIMCA analysis of plant canopy spectral data," Biosystems Engineering, vol. 171, pp. 143 – 154, 2018. [Online]. Available: https://doi.org/10.1016/j.biosystemseng.2018.04.019

[17] L. Li, X. Wei, H. Mao, and S. Wu, "Design and application of spectrum sensor for weed detection used in winter rape field," Transactions of the Chinese Society of Agricultural Engineering, vol. 33, no. 18, pp. 127–133, 2017. [Online]. Available: https://doi.org/10.11975/j.issn.1002-6819.2017.18.017

[18] M. Louargant, G. Jones, R. Faroux, J.-N. Paoli, T. Maillot, C. Gée, and S. Villette, "Unsupervised classification algorithm for early weed detection in row-crops by combining spatial and spectral information," Remote Sensing, vol. 10, no. 5, p. 761, 2018. [Online]. Available: https://doi.org/10.3390/rs10050761

[19] F. Lin, D. Zhang, Y. Huang, X. Wang, and X. Chen, "Detection of corn and weed species by the combination of spectral, shape and textural features," Sustainability, vol. 9, no. 8, p. 1335, 2017. [Online]. Available: https://doi.org/10.3390/su9081335

[20] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proceedings of the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, USA, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[21] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[22] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105, imageNet. [Online]. Available: https://dl.acm.org/citation.cfm?id=2999257

[23] D. C. Cireşan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "Flexible, high performance convolutional neural networks for image classification," in Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two, ser. IJCAI'11. AAAI Press, 2011, pp. 1237–1242. [Online]. Available: https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-210

[24] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Convolutional neural network committees for handwritten character classification," in 2011 International Conference on Document Analysis and Recognition, Sep. 2011, pp. 1135–1139. [Online]. Available: https://doi.org/10.1109/ICDAR.2011.229

[25] D. C. Ciresan, U. Meier, J. Masci, L. M. Gambardella, and J. Schmidhuber, "High-performance neural networks for visual object classification," CoRR, vol. abs/1102.0183, 2011. [Online]. Available: http://arxiv.org/abs/1102.0183

[26] D. C. Ciresan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," CoRR, vol. abs/1202.2745, 2012. [Online]. Available: http://arxiv.org/abs/1202.2745

[27] F. J. Knoll, M. Grelcke, V. Czymmek, T. Holtorf, and S. Hussmann, "CPU architecture for a fast and energy-saving calculation of convolution neural networks," pp. 10 335 – 10 335 – 9, 2017. [Online]. Available: https://doi.org/10.1117/12.2270282

[28] A. Milioto, P. Lottes, and C. Stachniss, "Real-time blob-wise sugar beets vs weeds classification for monitoring fields using convolutional neural networks," ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences, vol. IV-2/W3, pp. 41–48, 2017. [Online]. Available: https://doi.org/10.5194/isprs-annals-IV-2-W3-41-2017

[29] I. Sa, Z. Chen, M. Popović, R. Khanna, F. Liebisch, J. Nieto, and R. Siegwart, "weednet: Dense semantic weed classification using multispectral images and mav for smart farming," IEEE Robotics and Automation Letters, vol. 3, no. 1, pp. 588–595, Jan 2018. [Online]. Available: https://doi.org/10.1109/LRA.2017.2774979

[30] A. Milioto, P. Lottes, and C. Stachniss, "Real-time semantic segmentation of crop and weed for precision agriculture robots leveraging background knowledge in cnns," in 2018 IEEE International Conference on Robotics and Automation (ICRA), May 2018, pp. 2229–2235. [Online]. Available: https://doi.org/10.1109/ICRA.2018.8460962

[31] S. Biookaghazadeh, M. Zhao, and F. Ren, "Are FPGAs suitable for edge computing?" in USENIX Workshop on Hot Topics in Edge Computing (HotEdge 18). Boston, MA: USENIX Association, 2018.

[32] E. Nurvitadhi, G. Venkatesh, J. Sim, D. Marr, R. Huang, J. Ong Gee Hock, Y. T. Liew, K. Srivatsan, D. Moss, S. Subhaschandra, and G. Boudoukh, "Can FPGAs beat GPUs in accelerating next-generation deep neural networks?" in Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA '17. New York, NY, USA: ACM, 2017, pp. 5–14. [Online]. Available: http://doi.org/10.1145/3020078.3021740

[33] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, T. Czajkowski, S. D. Brown, and J. H. Anderson, "LegUp: An open-source high-level synthesis tool for FPGA-based processor & accelerator systems," ACM Trans. Embed. Comput. Syst., vol. 13, no. 2, pp. 24:1–24:27, Sep. 2013. [Online]. Available: https://doi.org/10.1145/2514740

[34] J. Cong, B. Liu, S. Neuendorffer, J. Noguera, K. Vissers, and Z. Zhang, "High-level synthesis for FPGAs: From prototyping to deployment," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 30, no. 4, pp. 473–491, April 2011. [Online]. Available: https://doi.org/10.1109/TCAD.2011.2110592

[35] H. Abdelkrim, S. Ben Othman, and S. Ben Saoud, "Reconfigurable SoC FPGA based: Overview and trends," in 2017 International Conference on Advanced Systems and Electric Technologies, Jan 2017, pp. 378–383. [Online]. Available: https://doi.org/10.1109/ASET.2017.7983723

[36] S. I. Venieris and C.-S. Bouganis, "fpgaConvNet: A Framework for Mapping Convolutional Neural Networks on FPGAs," in 2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM). Institute of Electrical and Electronics Engineers (IEEE), May 2016, pp. 40–47. [Online]. Available: https://doi.org/10.1109/FCCM.2016.22

[37] Q. Xiao, Y. Liang, L. Lu, and S. Y. and, "Exploring heterogeneous algorithms for accelerating deep convolutional neural networks on FPGAs," in 2017 54th ACM/EDAC/IEEE Design Automation Conference (DAC), June 2017, pp. 1–6. [Online]. Available: https://doi.org/10.1145/3061639.3062244

[38] J. H. Kim, B. Grady, R. Lian, J. Brothers, and J. H. Anderson, "FPGA-based CNN inference accelerator synthesized from multi-threaded C software," in 2017 30th IEEE International System-on-Chip Conference (SOCC), Sep. 2017, pp. 268–273. [Online]. Available: https://doi.org/10.1109/SOCC.2017.8226056

[39] K. Guo, L. Sui, J. Qiu, J. Yu, J. Wang, S. Yao, S. Han, Y. Wang, and H. Yang, "Angel-Eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA," IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 37, no. 1, pp. 35–47, Jan 2018. [Online]. Available: https://doi.org/10.1109/TCAD.2017.2705069

[40] K. Duan, W. Keerthi, S. Sathiya, S. Chu, K. Shirish, and A. Poo, "Multi-category classification by soft-max combination of binary classifiers," in Multiple Classifier Systems, T. Windeatt and F. Roli, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003, pp. 125–134, softmax. [Online]. Available: https://doi.org/10.1007/3-540-44938-8_13

[41] M. Courbariaux and Y. Bengio, "Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1," CoRR, vol. abs/1602.02830, 2016. [Online]. Available: http://arxiv.org/abs/1602.02830

[42] A. Y. Ng, "Feature selection, l1 vs. l2 regularization, and rotational invariance," in Proceedings of the Twenty-first International Conference on Machine Learning, ser. ICML '04. New York, NY, USA: ACM, 2004, pp. 78–. [Online]. Available: http://doi.acm.org/10.1145/1015330.1015435

[43] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," The Journal of Machine Learning Research, vol. 15, no. 1, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[44] L. Perez and J. Wang, "The effectiveness of data augmentation in image classification using deep learning," CoRR, vol. abs/1712.04621, 2017. [Online]. Available: http://arxiv.org/abs/1712.04621

[45] W. Tang, G. Hua, and L. Wang, "How to train a compact binary neural network with high accuracy?" in AAAI, 2017.

[46] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2324, Nov 1998, mNIST. [Online]. Available: https://doi.org/10.1109/5.726791

[47] A. Krizhevsky, "Learning multiple layers of features from tiny images," 2009, cIFAR-10.

[48] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," CoRR, vol. abs/1409.1556, 2014. [Online]. Available: http://arxiv.org/abs/1409.1556

[49] G. Huang, Z. Liu, and K. Q. Weinberger, "Densely connected convolutional networks," CoRR, vol. abs/1608.06993, 2016. [Online]. Available: http://arxiv.org/abs/1608.06993

[50] S. Zagoruyko and N. Komodakis, "Wide residual networks," CoRR, vol. abs/1605.07146, 2016. [Online]. Available: http://arxiv.org/abs/1605.07146

[51] N. Qian, "On the momentum term in gradient descent learning algorithms," Neural Networks, vol. 12, no. 1, pp. 145 – 151, 1999, momentum. [Online]. Available: https://doi.org/10.1016/S0893-60809800116-6

[52] G. Xavier and B. Yoshua, "Understanding the difficulty of training deep feedforward neural networks," in Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics, ser. Proceedings of Machine Learning Research, Y. W. Teh and M. Titterington, Eds., vol. 9. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256, glorot Initialization. [Online]. Available: http://proceedings.mlr.press/v9/glorot10a.html

[53] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in pytorch," 2017.

[54] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J.-s. Seo, and Y. Cao, "Throughput-optimized opencl-based fpga accelerator for large-scale convolutional neural networks," in Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, ser. FPGA '16. New York, NY, USA: ACM, 2016, pp. 16–25. [Online]. Available: http://doi.acm.org/10.1145/2847263.2847276

[55] C. Lammie, T. J. Hamilton, A. van Schaik, and M. Rahimi Azghadi, "Efficient FPGA Implementations of Pair and Triplet-Based STDP for Neuromorphic Architectures," IEEE Transactions on Circuits and Systems I: Regular Papers, 2018. [Online]. Available: https://doi.org/10.1109/TCSI.2018.2881753

[56] I. Hubara, M. Courbariaux, D. Soudry, R. El-Yaniv, and Y. Bengio, "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations," J. Mach. Learn. Res., vol. 18, no. 1, pp. 6869–6898, Jan. 2017. [Online]. Available: http://dl.acm.org/citation.cfm?id=3122009.3242044

[57] "Deep image: Scaling up image recognition," CoRR, vol. abs/1501.02876, 2015, withdrawn. [Online]. Available: http://arxiv.org/abs/1501.02876

[58] G. Mishra, Y. L. Aung, M. Wu, S. Lam, and T. Srikanthan, "Real-time image resizing hardware accelerator for object detection algorithms," in 2013 International Symposium on Electronic System Design, Dec 2013, pp. 98–102. [Online]. Available: https://doi.org/10.1109/ISED.2013.26

COREY LAMMIE (S'17) is currently pursuing a PhD in Computer Engineering at James Cook University (JCU), where he completed his undergraduate degrees in Electrical Engineering (Honours) and Information Technology in 2018. His main research interests include brain-inspired computing, Spiking Neural Networks (SNNs), Artificial Neural Networks (ANNs), and digital design using High Level Synthesis (HLS) and Hardware Descriptive Languages (HDLs).

He has served as a reviewer for the IEEE International Symposium on Circuits and Systems (ISCAS) and IEEE Transactions on Circuits and Systems II: Express Briefs.

ALEX OLSEN (S'17) is currently pursuing a PhD in Engineering and Related Technologies at James Cook University, Townsville, Australia, where he completed his Bachelor of Electrical Engineering and Bachelor of Mathematics in 2013 with Honors. From 2015 to 2018, he was a Lecturer and Tutor for the College of Science and Engineering at James Cook University. His primary fields of study are machine learning, image processing and robotics; with specific interest toward their application in agricultural practice.

TONY CARRICK (S'17) completed his Bachelor of Electrical Engineering (Honours) with James Cook University in 2018. His work has included research on diverse topics such as sensor integration for high power lasers, binarized neural networks, and most recently, using Neural Networks for tracking applications. He is currently involved with developing neural network architectures for object detection, whilst also assisting with part time research into neural networks and Precision Agriculture at James Cook University.

MOSTAFA RAHIMI AZGHADI (S'07–M'14) completed his PhD in Electrical & Electronic Engineering at The University of Adelaide, Australia, earning the Doctoral Research Medal, as well as the Adelaide University Alumni Medal in 2014. From 2012-2014, he was a visiting PhD student in the Neuromorphic Cognitive System group, Institute of Neuroinformatics, University and Swiss Federal Institute of Technology (ETH) Zurich, Switzerland.

He is currently a senior lecturer at the College of Science and Engineering, James Cook University, Townsville, Australia, where he researches neuromorphic engineering and brain-inspired architectures. Dr. Rahimi was a recipient of several national and international awards and scholarships such as Queensland Young Tall Poppy Science Award in 2017 and South Australia Science Excellence Awards in 2015. He serves as an associate editor of *Frontiers in Neuromorphic Engineering* and *IEEE Access*.