Article

Optimising Hardware Accelerated Neural Networks with Quantization and a Knowledge Distillation Evolutionary Algorithm

Robert Stewart [1], Pascal Bacchus [2], Andrew Nowlan [1], Quentin Ducasse [3] and Ekaterina Komendantskaya[1]

1 Mathematical and Computer Sciences, Heriot-Watt University, UK
2 INRIA Rennes, France
3 ENSTA Bretagne, France

Version December 6, 2020 submitted to Electronics

Abstract: This paper compares the latency, accuracy and hardware costs of neural networks compressed with our new multi-objective evolutionary algorithm called NEMOKD, and with quantization. We evaluate NEMOKD on Intel's Myriad X VPU processor, and quantization on Xilinx's programmable Z7020 FPGA hardware. Evolving models with NEMOKD increases inference accuracy by up to 82% at the cost of 38% increased latency, with throughput performance of 100-590 image frames-per-second (FPS). Quantization identifies a sweet spot of 3 bit precision in the trade-off between latency, hardware requirements and accuracy. Parallelising FPGA hardware implementations of quantized neural networks increases throughput from 6K FPS to 373k FPS.

Keywords: quantization, evolutionary algorithm, neural network, AI accelerator, FPGA, Movidius.

1. Introduction

Neural networks have proved successful for many domains including image recognition, autonomous systems and language processing. State-of-the-art models have an enormous number of parameters, making them highly computationally and memory intensive. For example, AlexNet [1] is a CNN consisting of 60 million parameters and 650k neurons with an architecture comprised of five convolutional layers, multiple max-pooling layers, three fully-connected layers and a final softmax layer. GPUs are often used to train and use neural networks because they can deliver the highest peak arithmetic performance for 32 bit floating point neural network inference compared with CPUs and FPGAs. At the time when the AlexNet model was proposed (2012), the network was too large to fit on a single GPU. This challenge was overcome by distributing the model two GPUs for training. The use of a 200+ Watt GPU for such purposes is becoming prohibitively expensive [2].

In recent years, a new class of hardware has emerged to significantly improve performance-per-Watt for deep learning. Accelerator devices such as the Intel Movidius Myriad X VPU [3] and the Coral/Google Edge TPU [4] accommodate deep learning workloads because they provide a trade off between compute performance and power consumption. The extreme on the hardware spectrum is programmable hardware like FPGAs, which provide extremely high throughput performance of fixed-point deep learning inference [5] for real-time applications like remote computer vision and automated stack market trading.

It is widely accepted that neural network models exhibit a high level of redundancy. Most parameters contribute little or nothing to the final output [6], and the precision of arithmetic calculations are unnecessarily precise [7]. Removing redundant bloat offer the opportunity of mapping sophisticated models to energy efficient devices. Methods for compressing neural networks include

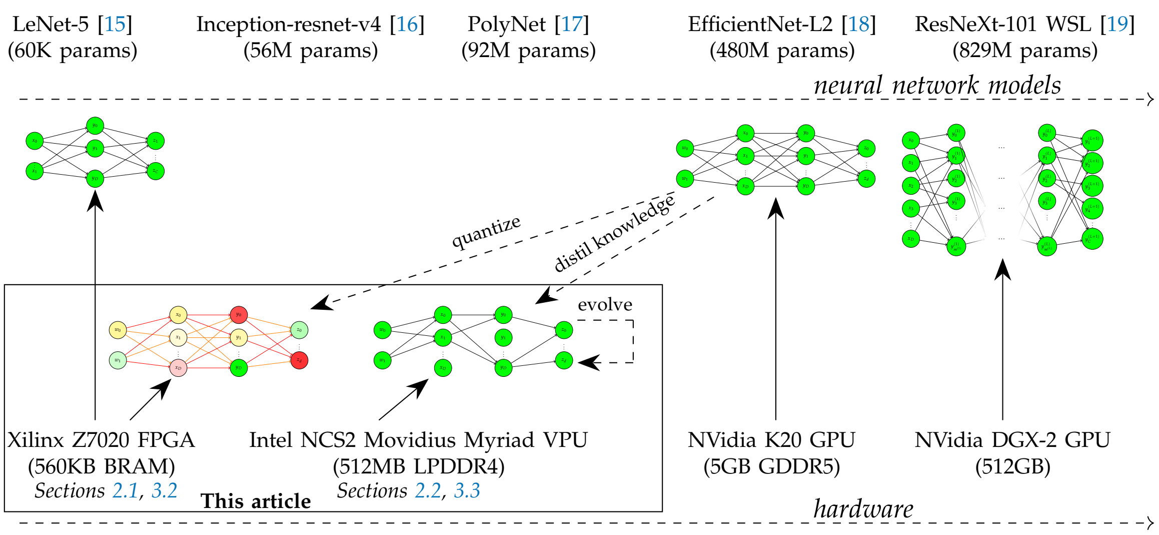


Figure 1. Meeting in the Middle: Compressing Neural Networks for Acceleration

precision reduction, redundant parameter/structure removal and transferring knowledge from large models to smaller models [8]. The larger of the two models is the *teacher*, the smaller distilled model is the *student*.

The aim of compression is usually to reduce the hardware footprint of a model to increase its inference throughput (decreasing its inference latency), without overtly affecting inference accuracy.

Compressing neural network can:

Speed up inference time Large neural network layers are memory bound, limited by memory bandwidth. This introduces latency that dominates execution time because most time is spent to bring data to processors rather than performing computation.

Improve energy efficiency It costs orders-of-magnitude more energy to access off-chip DDR memory compared to on-chip memory e.g. SRAM, BRAM and cache memory. Fitting weights into on-chip memories reduces frequency of energy inefficient off-chip memory accesses. Quantized fixed-point representations can significantly reduce energy costs [9], e.g. less than 5 Watts on FPGAs [10].

Reduce verification costs Recent SMT-based verification approaches aim to prove a neural network's robustness against adversarial attacks e.g. [11,12]. SMT solvers generally do not support non-linear arithmetic so activation functions must be linearised, which approximates a model for the purpose verification, rendering verification results unreliable. Quantizing activation functions can increase reliability of verifying neural networks robust [13], because it is the same model being verified and deployed. Moreover quantized models can be as robust against adversarial attack as their full precision baseline, possibly because quantization acts as a filter of subtle adversarial noise [14].

Neural network models vary hugely in their sizes, i.e. from 60 thousand parameters up to 900 million parameters. Figure 1 shows how compression such as quantization or student-teacher training can put relatively large models within reach of high throughput hardware accelerators.

This paper evaluates the accuracy, throughput and resource cost performance of two compression approaches applied to different sized models: (1) an *evolutionary algorithm to modify neural networks* targeting the Intel Movidius Myriad X VPU, and (2) *quantization* of fixed models targeting the Xilinx Z7020 FPGA.

Contributions

The paper makes the following contributions:

- A new framework called NEMOKD for hardware aware evolution of knowledge-distilled student models (Section 2.2).
- An evaluation of quantization based on inference accuracy, throughput, hardware requirements and training time on a programmable FPGA (Sections 3.2).
- An evaluation of NEMOKD showing its ability to minimise both latency and accuracy loss on the fixed VPU architecture (Section 3.3).
- A comparison of NEMOKD and quantization performance (Section 3.4).

2. Methodology

2.1. Quantization for FPGAs

Floating-point-quantized models permit individual parameters a range of exponent values. Larger exponent (higher precision) values can induce more computational overhead, leading to higher power consumption and longer compute times. Fixed-point quantized models use (usually smaller) fixed exponent values for all network parameters. This imposed restriction brings a range of benefits such as faster and more power efficient mathematical operations but can also potentially impact the model accuracy [20].

Quantization [21] shifts values from 32 bit floating point continuous values to reduced bit discrete values. In a neural network, weights between neurons and activiation functions can be quantized.

Binarization [22] is a special case of quantization that represents weights and/or activation function outputs with a single bit. These methods replace arithmetic operation with bit-wise operations, reducing the energy consumption and memory requirements.

Quantized neural networks can signifiantly outperform binarized neural networks and can compete with the accuracy of full precision models [21].

2.1.1. FINN Framework

Section 3.2 evaluates very low precision neural networks, quantizing precision from 32 bits to 1-8 bits to fit within the resource constraints of FPGAs. Xilinx's FINN quantization framework and FPGA backend is used in these experiments. FINN initially supported binarized neural networks [7], then was extended for quantized networks [23] and Long-Short Term Memory Neural Networks (LSTM) [24]. Our experiments in Section 3.2 uses FINN functionality from [23].

FINN employs quantization aware training at the Python level, before generating synthesisable C++ for hardware. The weights and activation functions during training in Python operate on floating point values but Python functions simulate quantization to limit weights and activation function outputs to discrete values permitted by the chosen quantization configuration. When generating hardware, the arithmetic precision of weights and activation functions in the C++ match the quantized bit widths simulated during training.

2.1.2. Weight Quantization for Training

FINN discretises the range of full precision values by rounding to a close neighbour to fixed point quantized values for weights. The *min* and *max* values for the quantization range are related to the quantization precision n, they are defined as:

$$max = 2 - \frac{1}{2^{n-2}} \qquad\qquad min = -2 + \frac{1}{2^{n-2}}$$

The quantization formula for $x \in [min; max]$ is shown in Equation 1.

Value	Precision (bits)							
	1	2	3	4	5	6	7	8
0.136	1	0	0	0.25	0.125	0.125	0.125	0.140625
0.357	1	0	0.5	0.25	0.375	0.375	0.34375	0.359375
0.639	1	1	0.5	0.75	0.625	0.625	0.625	0.640625
1.135	1	1	1	1.25	1.125	1.125	1.125	1.140625
2	1	1	1.5	1.75	1.875	1.9375	1.96875	1.984375

Table 1. Example of quantized weights

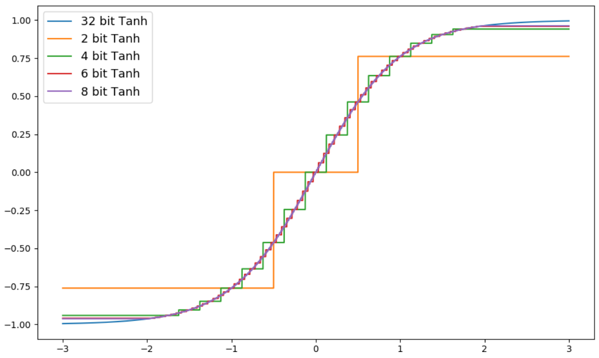


Figure 2. Hyperbolic tangent with different quantization configuration

$$QuantizeWeights(x) = \frac{\lfloor 2^n x + 2^{n-1} - 1 \rfloor}{2^{n-2}} - 2 + \frac{1}{2^{n-2}} \quad (1)$$

Table 1 shows examples of quantized values with $min = -2$ and $max = 2$ with $2^n - 1$ values in this interval. The values are all strictly positive but the quantization range is symmetric. The step between each quantized value is $\frac{1}{2^{n-2}}$. When n increases, the number of quantized values increase and we can obtain values close to the upper and lower bound of the interval.

2.1.3. Activation Function Quantization for Training

The quantization of activation functions works similarly to weight quantization. For the quantized hyperbolic tangent function $tanh(x) = \frac{e^x - e^{-x}}{e^x - + e^{-x}}$, the range of values in Table 1 is optimal because it has two asymptotes that goes towards -1 and 1, e.g. $tanh(2) = 0.964$. The saturation plateau of the activation function is almost attained. Figure 2 shows the shape of $tanh$ for different quantization precisions.

2.2. Neural Network Evolution

Pruning

Pruning eliminates redundant neural network parameters [20]. It is a common misconception that removing parameters from a neural network result in both (1) a linear decrease in memory footprint and (2) a linear decrease in inference time across all hardware architectures. It is therefore important to distinguish between these two objectives. It has been shown that for specific hardware configurations, removing parameters provides no latency improvements, and in some cases is detrimental [25,26]. Moreover hardware specific latency optimisations targeting one hardware architecture are not guaranteed to be optimal on different architectures [27].

Pruning can be performed at two different levels of granularity: either as fine-grained pruning of individual weight elements, or coarse-grained pruning of entire groups of elements, for example channels or filters. Pruning can significantly reduce the number of parameters without impacting inference accuracy, e.g. in [6] the effect on accuracy is negligible until nearly 90% of parameters have been pruned. The authors were able to reduce the AlexNet model from 68M parameters to 6.7M parameters without incurring any accuracy loss. A dynamic approach to fine-grained pruning in [28] proposes a splicing operation that allows connections to be recovered if they are later found to be important, as parameter importance may radically change once others have been pruned away.

Evolutionary Algorithms

Evolutionary deep learning approaches [29] have been proposed as an alternative training approach to stochastic gradient descent. However, due to the enormity of the search space for state-of-the-art neural networks that comprise millions of parameters, evolutionary algorithms often fail to discover optimal solutions.

Recent neuro-evolution techniques retain stochastic gradient descent and back propagation for training, before using evolutionary algorithms to search for optimal architectural configurations. Device-aware Progressive Search for Pareto-Optimal Neural Architectures [27] is a method of neural architecture search that has been shown to simultaneously optimise device-related objectives such as inference time and device-agnostic objectives such as accuracy. This search algorithm uses progressive search and mutation operators to explore the trade-offs between these objectives.

Applying this algorithm to problems on a range of different hardware devices from a NVIDIA Titan X GPU to a mobile phone with an ARM Cortex-A35, the authors were able to obtain higher accuracy and shorter inference times compared to the state-of- the-art CondenseNet [30]. Neural-Evolution with Multi-objective Optimisation (NEMO) [31] is an neural network optimisation algorithm. It is a machine learning technique that uses multi-objective evolutionary algorithms to simultaneously optimise both accuracy and inference time of neural networks by evolving their architecture [31].

2.2.1. NEMO with Knowledge Distillation for the VPU

To optimise neural networks for the VPU hardware, we use an approach that combines knowledge distillation and multi-objective evolutionary algorithms to minimise inference latency whilst also minimising accuracy loss. Knowledge distillation is a compression technique concerned with the transfer of knowledge from a large complex model or ensemble of models to smaller network architectures. Our methodology follows the Neuro-Evolution with Multi-objective Optimisation (NEMO) approach presented in [31].

Our NEMOKD framework (NEMO with Knowledge Distillation) incorporates knowledge distillation and profiles students based on their performance in early stages of training to reduce computational overhead.

The framework comprises two phases:

Phase 1: Knowledge Distillation For the same baseline student architecture and the same teacher model, NEMOKD iterates through different combinations of knowledge distillation hyper-parameters to identify the optimal hyper-parameters that yield the best student accuracy after.

Phase 2: Model Evolution NEMOKD then iteratively evolves four hyper-parameters (below) of a chosen baseline student architecture. The NEMOKD framework profiles students based upon their performance in early stages of training.

We employ the Non-Dominated Sorting Genetic Algorithm version II (NSGAII) [32] to facilitate hardware-aware evolutionary multi-objective optimisation, targeting the Myriad X VPU. Our approach extends [31] with three extensions:

1. Knowledge distillation replaces standard training in the learning phase of the evaluation procedure.

Device	Model		Dataset	Section
Xilinx Z7020 FPGA	3 layer fully connected MLP		MNIST	3.2.1, 3.2.2
(quantization)	3 layer fully connected MLP		FASHION-MNIST	3.2.3
	Teacher	*Student*		
Intel Movidius Myriad X VPU	MobileNetV2	FlexStudent	CIFAR10	3.3
(model evolution)	Resnet32x4	FlexStudent	CIFAR100	3.3
	Resnet32x4	Resnet8x4	CIFAR100	3.3

Table 2. Quantization and Model Evolution Experiments

2. To conserve time and computational resources in the learning phase, partial training is provided as opposed to training each model fully.
3. Inference test error and latency are measured on the Myriad X VPU device, which feeds into the evolutionary NSGAII algorithm.

Model mutations with NSGAII are both fine and coarse grained. Mutation in our NEMOKD framework modifies four hyper-parameters:

1. The number of convolutional layers.
2. The number of Fully Connect layers.
3. The number of output channels.
4. The number of fully-connected neurons.

The evolutionary process mutates these hyper-parameters to perform cross-over and mutation to members of the selected population before benchmarking, ranking and selecting children module solutions before iterating this new population. The evolution based optimisation in NEMOKD is *hardware aware*, because latency and accuracy benchmarking is performed on the Myriad X VPU device.

3. Evaluation

3.1. Hardware Platforms

This section evaluates quantization for programmable hardware, and our NEMOKD evolutionary algorithm for fixed AI hardware. The dataset and neural network model for the experiments are shown in Table 2.

For the *programmable hardware* experiments we target the mid-range Xilinx Zynq Z7020 140$mm \times$ 87mm FPGA on the PYNQ-Z2 development board which uses ≈13.8W energy. This FPGA has 53k Lookup Tables (LUT), 106k Flip Flops (FF) and 560KB of Block RAM (BRAM) memory. Of the 64 quantized neural networks, only four actually fit on this FPGA, validating the need for aggressive compression approaches such as quantization, on small/medium sized FPGA devices.

For the *fixed hardware* experiments we use a USB-based Intel Movidius NCS2 accelerator using a 72.5$mm \times$ 27mm Visual Processing Unit (VPU) which uses ≈1.5W energy. The NCS2 comprises dedicated accelerators with the 16 programmable 128-bit VLIW Vector Processors optimised for processing highly parallel workloads. The device can compute up to 1 Tera Operations Per Second (TOPS). The centralised 2.5MB of on chip memory facilitated by the intelligent memory fabric enables memory access latencies of 400 GB/s and reduces the requirements for more costly off-chip data transfer [33]. The USB Neural Compute Stick 2 (NCS2) has 512MB of LPDDR4 memory.

3.2. Quantization Results

The following sections explores the design space granted by FINN's ability to independently quantize weights and activation functions of a Multilayer Perceptron (MLP) network with three Fully-Connected (FC) layers. We create 64 quantized models from a baseline model by independently and exhaustively varying the bit-widths of weights and activation functions from 1 to 8.

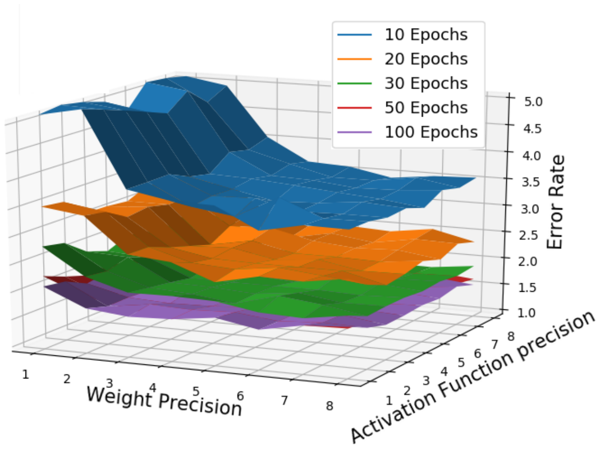


Figure 3. Accuracy of QNNs with Increasing Training

3.2.1. Hardware/Accuracy Trade Offs

The training is done using 50000 images from the MNIST dataset. A validation dataset of 10000 images is then used to minimise overfitting. Accuracy is measured using a testing dataset, to test how well the model generalises to new data. FINN's backend converts the model to a binary weight file and a synthesisable C++ implementation for hardware. For 64 neural network quantization configurations, the evaluation in this section measures:

1. *Absolute* accuracy and hardware resource costs of the 64 quantized neural networks.
2. *Relative* performance comparison of accuracy and hardware resource costs.

Absolute Accuracy Performance

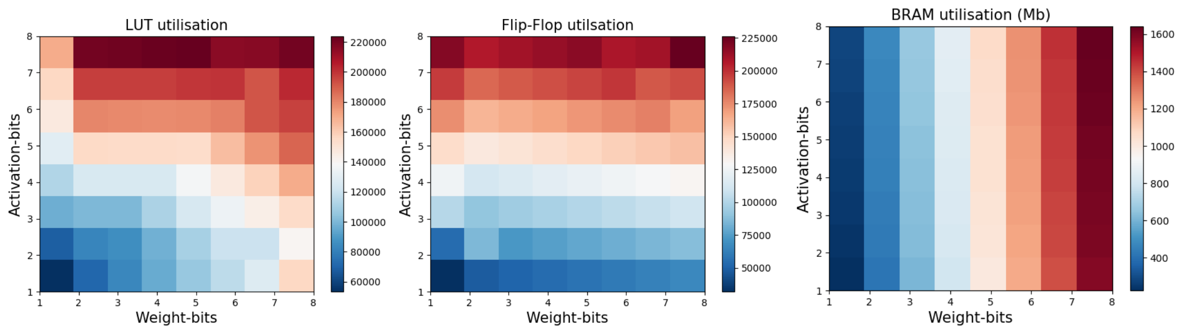
Each of the 64 neural networks is labelled with a quantized weight W-X and quantized activation function A-Y with $X, Y \in [1;8]$. Accuracy is measured after 10, 20, 30, 50 and 100 epochs.

Figure 3 plots the inference error rate for each of the 64 quantized neural networks after training with 10, 20, 30, 50 and 100 epochs. Using 1-3 bits weights has a noticeable effect on accuracy, i.e. between 3.9%-4.7% dropping down to below 3.7% using 4 bits or more. Training further with 40-100 epochs shifts the noticeable accuracy boundary to just 1 bit weight, meaning that with enough training, 2 bit weights achieves almost the same inference accuracy as 3-8 bit weights. The quantization of activation functions has a steady impact on accuracy, i.e. higher precision activation functions result in better accuracy, however, this is not as dramatic as the impact that quantized weight precision has on accuracy. With increased training time, the accuracy performance flattens, where absolute difference in accuracy between the best and worst quantization configuration greatly diminishes. Also, we observe a major gap between 1 and 2 bit weights versus 3-8 bit weights, especially for 10 and 20 epochs. Training beyond 40 epochs allows weights to be quantized from 3 to 2 bits without noticeable accuracy loss.

Absolute Resource Utilisation Performance

Figure 4 shows the trade-off between quantized precision and hardware resource use. The X axis is the number of bits for weights, the Y axis is the number of bits for the activation functions. The colour in the heat maps represents the relative measurement of the respective performance metric compared to the other 63 models.

Figure 4a shows that both weight precision and activation function precision contribute evenly to LUTs costs. Figure 4b shows that the precision of activation functions determines FF costs. While FFs and LUTs can store small amounts of data, BRAMs have greater storage capacity and are used by hardware synthesis tools for larger data structures such as arrays. Figure 4c shows that BRAM consumption is determined exclusively by weight precision.



(a) LUTs **(b)** FFs **(c)** BRAMs

Figure 4. Hardware Resources Required for 64 Quantized Neural Networks

Metric	Relative performance worst	best
Accuracy loss	2.07%	1.52%
BRAM	1643	224
Flip Flops	226282	31954
Look Up Tables	223910	53336

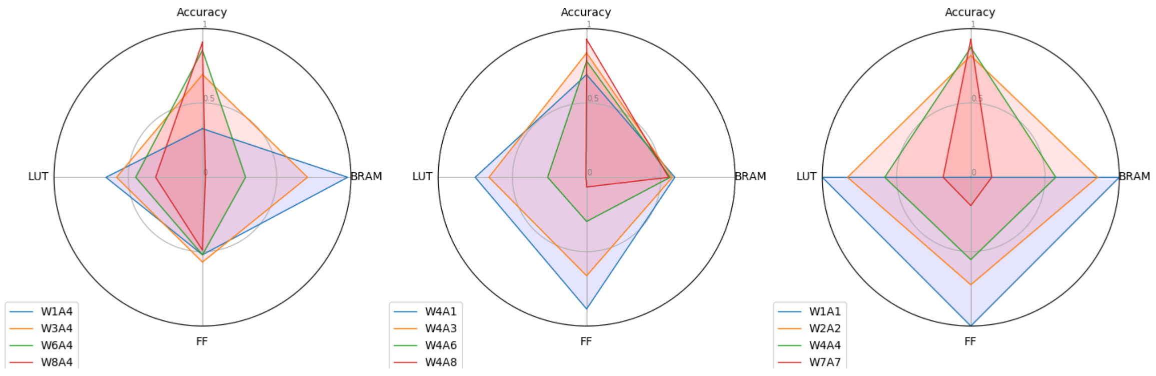
Table 3. Relative Performance for Radar Plots in Figure 5

3.2.2. Relative Quantization Performance

Table 3 gives the best and worst relative performance numbers for the 64 quantized neural networks. The three radar plots in Figure 5 represents different quantized neural network configurations, comparing accuracy and resource use (LUTs, FFs and BRAMs) performance relative to the best and values in Table 3. Each metric defines one branch in a radar chart. The three precision variations in Figure 5 are:

1. Weight oriented distribution (Figure 5a) increases the weight precision and keeps the activation function constant at 4 bits, i.e. W1-A4, W3-A4, W6-A4 and W8-A4.
2. Activation oriented distribution (Figure 5b) increases the activation function precision and keeps the weight precision constant at 4 bits, i.e. W4-A1, W4-A3, W4-A6 and W4-A8.
3. Linear distribution (Figure 5c) increases both the weight and activation function precision across the diagonal from the heat maps in Figure 4, i.e. W1-A1, W2-A2, W4-A4 and W7-A7.

The radar plots compare the relative performance of these quantization configurations. Their scores are normalised between scores of 0 and 1. The model with the highest accuracy is plotted



(a) Weight oriented distribution **(b)** Activation oriented distribution **(c)** Linear distribution

Figure 5. Radar charts for different quantization configurations

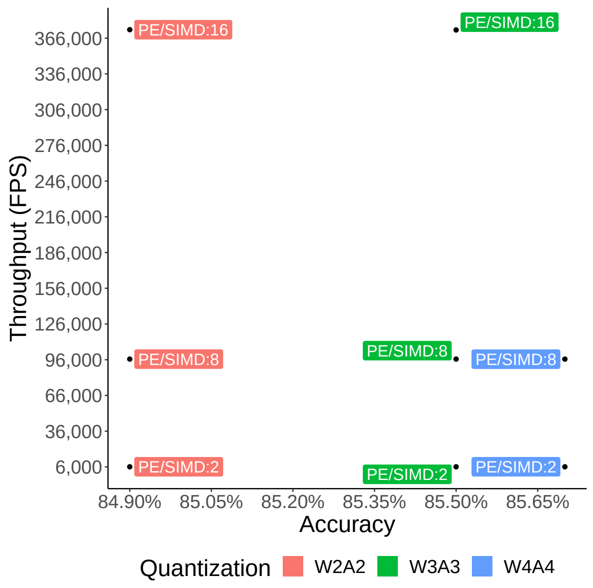


Figure 6. Throughput and Accuracy Performance of Parallel FPGA Designs for FASHION-MNIST

outermost in the radar plot in the *Accuracy* dimension whereas the models with lowest accuracy is plotted at the centre point. Likewise, the neural network using the fewest BRAMs is plotted outermost for the BRAM dimension, and the same for LUTs and FFs.

When activation functions are set to 3 bits, increasing weights from W1 to W3 causes the greatest relative accuracy score improvement (Figure 5a). When weights are fixed at 4 bits, all accuracy scores are in the top half, with increases of activation function precision costing significantly more LUT and FF resources, with BRAM costs largely the same (Figure 5b). Scaling both precision linearly has an equal impact on FF, LUT and BRAM scores, yet their accuracy score are all in the top quartile when weights are 2-8 bits (Figure 5c). In summary, if top-half relative accuracy performance is the goal, the most important constraint is 2+ bits for representing weights.

The importance of the trade-offs is highlighted by the fact that most of the neural networks do not fit on the target device (Xilinx Zynq Z7020). It has 280 BRAMs and only 7 of the networks meet this constraint, and 106400 FFs with 22 of the networks within this constraint.

3.2.3. Accuracy/Throughput Trade Offs

FINN supports parallelisation on a layer-by-layer basis. The amount of parallel hardware resources used to implement each layer of a neural network is user definable. Parallelism is controlled with two settings: (1) the number of hardware processing elements (PE) to processing each output channel, and (2) the number of input channels processed within one clock cycle (SIMD) [23]. Using more parallel hardware for a layer shortens the layer's clock cycle latency, at the cost of increased hardware requirements. If the layer is on the critical path, i.e. is has the highest latency cost, then parallelisation of that layer should shorten overall latency thereby increasing throughput.

The throughput evaluation uses a multi-layer perceptron with three fully connected layers with the FASHION-MNIST dataset. Each quantized model is tested for accuracy and throughput on the Xilinx Z7020 FPGA on the PYNQ-Z2 board. Each model is trained with 40 epochs. The results compare:

1. Inference accuracy.
2. Frames-Per-Second (FPS) image throughput.
3. Quantization configurations W2A2, W3A3 and W4A4.
4. The parallelism degree for PE and SIMD for all layers, setting both at 2, 8 then 16.

Figure 6 shows throughput results. The model with 2-bit precision achieves 84.9% accuracy. Increasing to 3-bit and 4-bit precision increases accuracy to 85.5% and 85.7% respectively. For 2-4 bit models, setting PE/SIMD to 2 achieves a throughput of 6k FPS. Increasing these parallelism parameters to 8 and 16 increases throughput to 96k and 373k FPS respectively. The W4A4 quanitzed model does not fit within the Xilinx Z7020 FPGA's available resources when PE/SIMD is 16, and hence is not shown inFigure 6. Increasing parallelism does not affect accuracy because each time it is the same model, just implemented with more parallel hardware.

3.2.4. Quantization Results Discussion

The sweet spot in the quantization design space for the MNIST and FASHION-MNIST datasets is about 3 bit weights and 3 bit activation functions. Beyond 3 bit quantization and with enough training, there is no significant improvement to accuracy performance. This confirms results in [24]. Our methodology for evaluating the trade-off between accuracy, throughput and hardware efficiency is similar to [34]. We extend that work by also measuring the impact of varying training of quantized models, and a more fine grained benchmark suite measuring weight precision independently of activation function precision.

In summary, our quantization experiments show:

- LUT and FF resources increase with increased activation function precision, because increasing arithmetic calculation complexity increases the number of required processing units.
- BRAM increases with increased weight precision, because weight parameters are stored in BRAM memories.
- Inference accuracy is highest with higher precision, i.e. least aggressive quantization. The biggest improvement in accuracy with a 1 bit increment is switching from 1 to 2 bits weight precision.
- With enough training beyond 50 epochs, 2 bit precision achieves almost the same inference accuracy as 3-8 bit precision.
- Increasing the parallelisation of hardware neural network implementations significantly increases throughput performance, from 6.1k FPS to 373k FPS.
- The optimal configuration with two objectives of throughput and parallelism is the W3A3 model with PE/SIMD parallelism parameters set to 16, achieving 373k FPS and 85.5% respectively.

3.3. NEMOKD Results

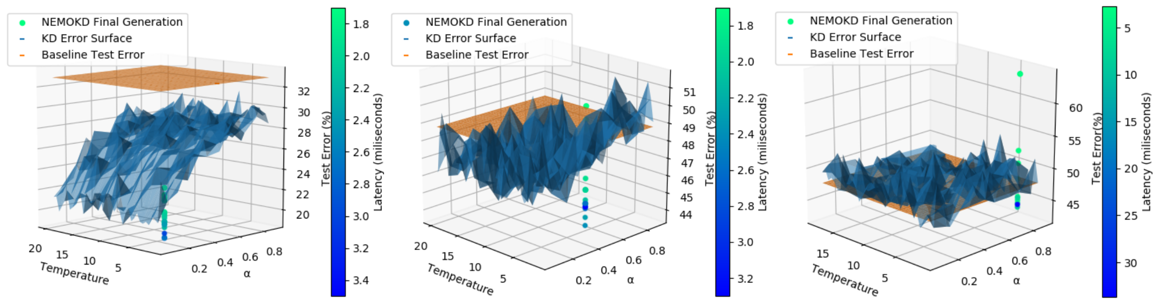
For the NEMOKD experiments in this section, we use two student models as our solution space for hyper-parameter evolution:

1. *FlexStudent*, a model that we have constructed with a simple five layer model to provide a starting point for the NEMOKD evolution process (Section 2.2.1). A similar model performs well as a student architecture on the CIFAR10 dataset [35].
2. A version of the *Resnet8x4* architecture, modified to enable the NEMOKD hyper-parameter evolutionary process.

Our NEMOKD framework is measured with three benchmarks:

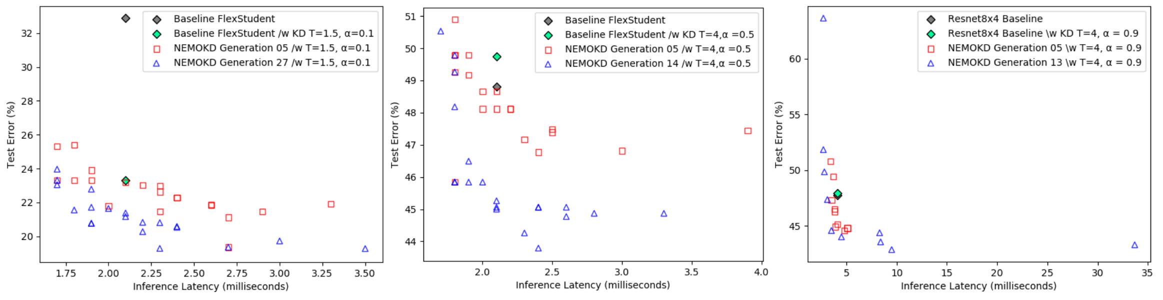
1. The *MobileNetV2* model distilled into a *FlexStudent* student model with the CIFAR10 dataset.
2. The *Resnet32x4* model distiller into a *FlexStudent* student model with CIFAR100.
3. The *Resnet32x4* model distilled into a *Resnet8x4* student model with CIFAR100. For this experiment, the number of layers remained fixed.

The experiments use 30 epochs for knowledge distillation, and 30 epochs for NSGAII model evolution. For our pruning benchmarks we use Platypus [36] for multi-objective optimisation, RepDistiller [37] for knowledge distillation, and OpenVino's Python API to execute trained exported PyTorch models on the NSC2 device.



(a) FlexStudent-MobileNetV2 **(b)** FlexStudent-Resnet32x4 **(c)** Resnet8x4-Resnet32x4

Figure 7. Knowledge Distillation Parameter Search



(a) FlexStudent-MobileNetV2 **(b)** FlexStudent-Resnet32x4 **(c)** Resnet8x4-Resnet32x4

Figure 8. Student Latency and Accuracy Performance

3.3.1. Knowledge Distillation Parameter Search

Figure 7 shows knowledge distillation error with 30 epochs. It illustrates how different combinations of knowledge distillation parameters affect the accuracy of the baseline model after 30 epochs. The α value determines how much the distillation loss and student loss contribute to the overall loss e.g. if $\alpha = 0.5$, then both terms in the knowledge distillation loss function are weighted evenly. The SoftMax function in the distillation loss term is parametrised by the temperature. This softens the output distribution revealing extra information about which classes the model found most alike. The blue surface illustrates the error rate of the baseline model with respect to different combinations of knowledge distillation hyper-parameters. The orange plane indicates the baseline test error performance without knowledge distillation. On this dataset, any choice of knowledge distillation hyper-parameters provides a significant increase in accuracy over the baseline model.

Figure 7b shows that some combinations of the knowledge distillation parameters have a negative effect on the accuracy of the baseline model. We observe that this method produces better accuracy than can be obtained by distilling knowledge into the baseline model, once again at the expense of latency.

In Figure 7c, the majority of combinations of knowledge distillation hyper-parameters have a negative impact on the baseline model accuracy, though certain combinations do provide improvements as shown in Figure 7c. In this case, no major trends are observed with respect to the individual hyper-parameters. We observe that this method, once again, produces better accuracy than can be obtained by distilling knowledge into the baseline model with 30 epochs of training.

The FlexStudent-MobileNetV2 experiment (Figure 7a) is similar to Figure 7b, but rather than CIFAR100 it uses the simpler CIFAR10 dataset. In this case, every combination of knowledge distillation hyper parameters provide significant improvement over baseline.

3.3.2. Efficacy of NEMOKD Evolution

Figure 8 shows the latency and accuracy performance of student models after 30 epochs for student models. As with the quantization experiments, accuracy is measured using a testing dataset.

Figure 8 illustrates the population at two distinct generations of the evolutionary process, in addition to the baseline architecture from which all the students have evolved.

Figure 8a shows FlexStudent-MobileNetV2 student performance. The chosen knowledge distillation hyper-parameters for this experiment have greatly increased the accuracy of the baseline model. With 30 epochs of training, many students in the final generation have evolved to attain a better accuracy than the baseline model but with the same or better latency. The same is also true of the baseline model trained with knowledge distillation. The most accurate students, however, have larger latency values with respect to the baseline model. The best trade-off model evolved five convolutional layers with a relatively small number of output channels and just two fully connected layers with a low to moderate number of neurons.

Figure 8b shows FlexStudent-Resnet32x4 student performance with the CIFAR100 dataset, and employs a larger teacher model: Resnet32x4. Student models evolve from the same baseline FlexStudent as the experiment in Figure 8a. Figure 8b illustrates the population at two distinct generations of the evolutionary process, in addition to the baseline architecture from which all the students have evolved. Interestingly, the combination of knowledge distillation hyper-parameters we chose for this experiment have had a negative impact on the accuracy of the baseline model. However, the evolved students appear to have adapted their architecture to accommodate these parameters, resulting in student models with significant accuracy improvements for the same inference latency. The best accuracy produced by the NEMOKD algorithm is obtained by an architecture with a higher latency. In contrast to Figure 8a, every student model in the final generation evolved to have the same layer structure as the baseline model.

Figure 8c shows Resnet8x4-Resnet32x4 student performance with the CIFAR100 dataset. It differs from Figures 8a and 8b in two ways: (1) a different evolutionary starting point is used for the student (ResNet8$\times$4); and (2) the layers of our flexible ResNet model are fixed, only the output channels of the convolutional layers are modified in the evolutionary process.

3.4. Discussion

Quantization for FPGAs

Our quantization experiments (Section 3.2) use the quantization scheme implemented in Xilinx's FINN framework. Developing compression algorithms for embedded devices is a research area of its own, e.g. a dynamic precision data quantization algorithm in [38], performed layer-by-layer from a corresponding floating point CNN, with the goal of improving bandwidth and resource utilisation. Other compression approaches are focused on specific goals e.g. reducing power consumption, or target specific hardware e.g. GPUs or FPGAs, or target specific domains or even specific application algorithms.

Device Specific Quantization Recent work explores the performance trade-offs between reduced precision of neural networks and their speed on GPUs, e.g. performance aware pruning can lead to 3-10 times speedups [39]. Multi-precision FPGA hardware for neural networks significantly reduces model sizes, which in [40] enables an ImageNet network to fit entirely on-chip for the first time, significantly speeding up throughput. Another recent study [24] measures the hardware cost, power consumption, and throughput for a High Level Synthesis extension of FINN that supports Long Short-Term memory (LSTM) models on FPGAs. [41] proposes a design flow for constructing low precision, low powered FPGA-based neural networks with a hybrid quantization scheme. [42] shows that resource-aware model analysis, data quantization and efficient use of hardware techniques can be combined to jointly map binarized neural networks to FPGAs with dramatically reduced resource requirements whilst maintaining acceptable accuracy.

Domain Specific Quantization Some quantization methods target specific algorithms, e.g. a resource-aware weight quantization framework for performing object detection in images [43].

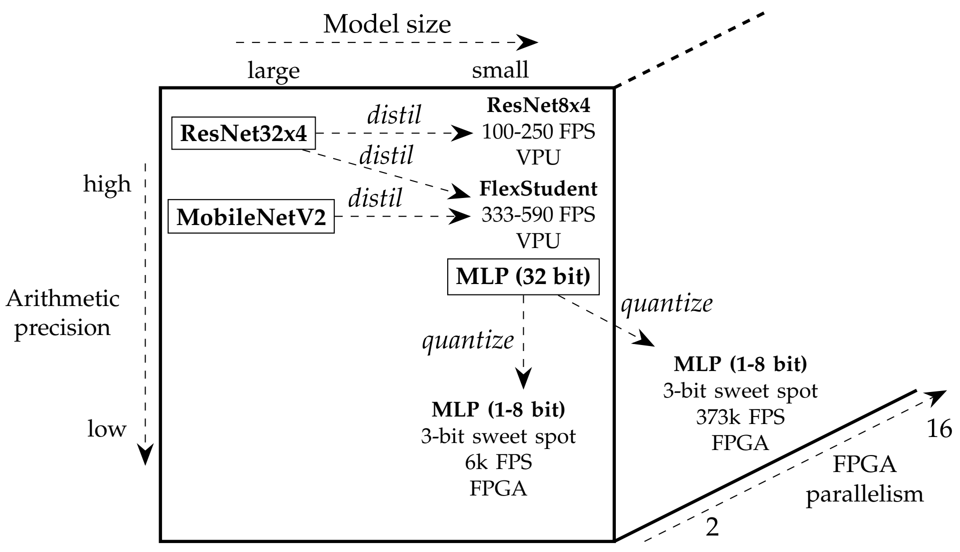


Figure 9. Varying Precision and Model Archiectures

NEMO with Knowledge Distillation for the VPU

Knowledge distillation parameters for the NEMOKD experiments (Section 3.3) greatly increase the accuracy of the baseline model. With 30 epochs of training, many students in the final generation evolve to attain a better accuracy than the baseline model but with the same or better latency. The most accurate students, however, have larger latency values with respect to the baseline model. The best trade-off model evolved five convolutional layers with a relatively small number of output channels and just two fully connected layers with a low to moderate number of neurons.

Our NEMOKD approach can significantly increase inference accuracy at a modest expense of latency. The method consistently provides higher accuracy students than could be obtained through an exhaustive knowledge distillation parameter search with the baseline model, irrespective of the choice of knowledge distillation hyper-parameters. This highlights the importance of the student's architecture in the knowledge distillation process. Evolving the students appeared to enable the model to adapt and accommodate an arbitrary choice of knowledge distillation hyper-parameters, even if the choice was initially detrimental to the accuracy of the baseline model.

3.4.1. Comparing Quantization and NEMOKD

The quantization and NEMOKD results are shown in Figure 9. Both compression approaches start from baseline models of *ResNet32x4* and *MobileNetV2* (for NEMOKD), and a 32 bit Multi-Layer Perceptron model (for quantization). Quantization reduces the arithmetic precision without changing a model's architecture, i.e. the number of hidden layers and number of neurons are unchanged. In contrast, the NEMOKD framework changes the model's architecture whilst leaving arithmetic precision unchanged. Typically, 30 image FPS throughput is considered real-time computer vision performance [44]. Quantization and the NEMOKD framework both achieve real-time image processing – 590 FPS on the VPU and 373k FPS on the FPGA.

4. Conclusions and Future Work

4.1. Conclusion

This paper explores two optimisation approaches for neural networks for programmable hardware and a fixed AI processor: (1) quantization precision of fixed models, and (2) evolving hyper-parameters of student models after knowledge distillation. There is a sweet spot of 3 bit quantization in the trade-off between latency, hardware requirements and accuracy. Parallelising hardware implementations of neural networks increases FPS from 6k to 373k. Evolving student models increases inference accuracy by up to 82% at the cost of 38% increased latency. The lowest inference latencies were 1.7ms for the FlexStudent-MobileNetV2 student, 1.4ms for the FlexStudent-Resnet32x4 student and 2ms for the Resnet8x4-Resnet32x4 student, i.e. a throughput of 100-590 FPS.

4.2. Future Work

Larger datasets and models

Our experiments use four datasets: MNIST and FASHION-MNIST for quantization, and CIFAR10 and CIFAR100 for NEMOKD. The quantization experiments are based on a five layer fully-connected network and the NEMOKD experiments use three student models. More work is required to scale accuracy-preserving compression methods to real world computer vision applications e.g. from 28×28 MNIST and FASHION-MNIST images, and 32×32 CIFAR10 and CIFAR100 images, to much higher dimensions such as 400×150 road lane detection images for autonomous driving [45]. Scaling compressing experiments to (1) deeper models with tens/hundreds of hidden layers, and (2) datasets with thousands of classes e.g. ImageNet, would be an intermediate step in that direction.

Automating Compression

Our quantization benchmarks were exhaustive in the design space of 1-8 bits for activation functions and weight values. The quantization was homogeneous across the entire network each time, i.e. each quantization configuration applied to all parameters. The FINN API supports per-layer activation function and weights precision, and the Brevitas framework [46] in FINN allows layer-by-layer clock cycle profiling. This opens up the opportunity for automating layer-by-layer quantization methodologies e.g. [47].

When using evolutionary algorithms with knowledge distillation for larger datasets and models, enabling more parameters to be the subject of mutation throughout the evolutionary process could prove beneficial in automating search for optimal compressed models. Recent teacher-student methods [48] outperform knowledge distillation in a wide range of problems. Designing a flexible student model that accommodates both evolution and more complex distillation methods would be considerably more challenging, but given the positive results we report for NEMOKD we believe this would be important future work.

Performance Portability of Compressed Models

The two compression methods in this paper were tested on one hardware platform each. Our NEMOKD approach is hardware-aware, since the multi-objective optimisation phase is measured on the Myriad X VPU device. Evolving the same initial model with the goal of minimising latency and accuracy loss may produce quite different models for different devices due to different memory latencies, cache size and the number of parallel processing elements on each device. For quantization, the amount of on-chip BRAM memory ranges from 0.5MB to 8MB for different FPGA devices, meaning aggressive quantization and binarization is needed for low-end devices, necessitating auto-tuning of model precision to be device specific.

Combining Knowledge Distillation with Quantisation

Previous work shows that *combining* compression methods can achieve superior performance compared with using them in isolation, e.g. combining pruning and knowledge distillation [26]. The approach in [49] shows that distilling knowledge to shallower quantized architectures can achieve accuracy comparable with state-of-the-art full-precision models. There are other compression methods such as weight sharing [50] to consider for hybrid compression. A complete study of neural network compression approaches is in [20].

More work is required to evaluate these hybrid neural network compression techniques at the scale of state-of-the-art real world problems. Not only may hybrid methods achieve superior throughput performance and energy efficiency, reducing precision and removing unimportant redundancy at scale may make verification of real-world deep learning models possible.

Author Contributions: The individual contributions are as follows. Robert Stewart: conceptualization; funding acquisition; investigation; project administration; resources; supervision; writing–original draft; writing–review and editing. Pascal Bacchus: data curation 3.2; formal analysis 3.4; software 2.1; visualisation 3.2; writing–original draft preparation 2.1 and 3.2. Quentin Ducasse: data curation 3.2.3; software 2.1; visualisation 3.2.3; writing–original draft preparation 3.2.3. Andrew Nowlan: data curation 3.3; formal analysis 3.3; software 2.2; visualisation 3.3; writing–original draft preparation 2.2 and 3.3. Ekaterina Komendantskaya: funding acquisition; supervision; writing–review and editing.

Funding: This research was funded by EPSRC project *Border Patrol: Improving Smart Device Security through Type-Aware Systems Design (EP/N028201/1)*; National Cyber Security Center, UK, Grant *SecConn-NN: Neural Networks with Security Contracts—towards lightweight, modular security for neural networks.*

Conflicts of Interest: The authors declare no conflict of interest.

1. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems 25: 26th Annual Conference on Neural Information Processing Systems 2012. Proceedings of a meeting held December 3-6, 2012, Lake Tahoe, Nevada, United States, 2012, pp. 1106–1114.
2. Lu, D. Creating an AI can be five times worse for the planet than a car, 2019. New Scientist.
3. Intel. Intel® Movidius™ Vision Processing Units (VPUs). https://www.intel.com/content/www/us/en/products/processors/movidius-vpu.html.
4. Coral.; Google. Edge TPU: Google's purpose-built ASIC designed to run inference at the edge.
5. Véstias, M.P.; Neto, H.C. Trends of CPU, GPU and FPGA for high-performance computing. FPL 2014, Munich, Germany, 2-4 September, 2014. IEEE, 2014, pp. 1–6.
6. Han, S.; Pool, J.; Tran, J.; Dally, W.J. Learning both Weights and Connections for Efficient Neural Network. Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada, 2015, pp. 1135–1143.
7. Umuroglu, Y.; Fraser, N.J.; Gambardella, G.; Blott, M.; Leong, P.H.W.; Jahre, M.; Vissers, K.A. FINN: A Framework for Fast, Scalable Binarized Neural Network Inference. FPGA 2017, Monterey, CA, USA, February 22-24, 2017. ACM, 2017, pp. 65–74.
8. Hinton, G.E.; Vinyals, O.; Dean, J. Distilling the Knowledge in a Neural Network. *CoRR* **2015**, *abs/1503.02531*, [1503.02531].
9. Chen, T.; Du, Z.; Sun, N.; Wang, J.; Wu, C.; Chen, Y.; Temam, O. DianNao: a small-footprint high-throughput accelerator for ubiquitous machine-learning. ASPLOS 2014, Salt Lake City, UT, USA, March 1-5, 2014. ACM, 2014, pp. 269–284.
10. Park, J.; Sung, W. FPGA based implementation of deep neural networks using on-chip memory only. ICASSP 2016, Shanghai, China, March 20-25, 2016. IEEE, 2016, pp. 1011–1015.
11. Katz, G.; Barrett, C.W.; Dill, D.L.; Julian, K.; Kochenderfer, M.J. Reluplex: An Efficient SMT Solver for Verifying Deep Neural Networks. Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I. Springer, 2017, pp. 97–117.
12. Liu, C.; Arnon, T.; Lazarus, C.; Barrett, C.W.; Kochenderfer, M.J. Algorithms for Verifying Deep Neural Networks. *CoRR* **2019**, *abs/1903.06758*, [1903.06758].
13. Kokke, W.; Komendantskaya, E.; Kienitz, D.; Atkey, R.; Aspinall, D. Neural Networks, Secure by Construction: An Exploration of Refinement Types. Asian Symposium on Programming Languages and Systems (APLAS). Springer, 2020.
14. Duncan, K.; Komendantskaya, E.; Stewart, R.; Lones, M.A. Relative Robustness of Quantized Neural Networks Against Adversarial Attacks. 2020 International Joint Conference on Neural Networks, IJCNN 2020, Glasgow, United Kingdom, July 19-24, 2020. IEEE, 2020, pp. 1–8.
15. Lecun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE* **1998**, *86*, 2278–2324.

16. Szegedy, C.; Ioffe, S.; Vanhoucke, V.; Alemi, A.A. Inception-v4, Inception-ResNet and the Impact of Residual Connections on Learning. Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence, February 4-9, 2017, San Francisco, California, USA. AAAI Press, 2017, pp. 4278–4284.
17. Zhang, X.; Li, Z.; Loy, C.C.; Lin, D. PolyNet: A Pursuit of Structural Diversity in Very Deep Networks. 2017 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017, Honolulu, HI, USA, July 21-26, 2017. IEEE Computer Society, 2017, pp. 3900–3908.
18. Tan, M.; Le, Q.V. EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks. Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA. PMLR, 2019, pp. 6105–6114.
19. Mahajan, D.; Girshick, R.B.; Ramanathan, V.; He, K.; Paluri, M.; Li, Y.; Bharambe, A.; van der Maaten, L. Exploring the Limits of Weakly Supervised Pretraining. *CoRR* **2018**, *abs/1805.00932*.
20. Wang, E.; Davis, J.J.; Zhao, R.; Ng, H.; Niu, X.; Luk, W.; Cheung, P.Y.K.; Constantinides, G.A. Deep Neural Network Approximation for Custom Hardware: Where We've Been, Where We're Going. *ACM Comput. Surv.* **2019**, *52*, 40:1–40:39.
21. Hubara, I.; Courbariaux, M.; Soudry, D.; El-Yaniv, R.; Bengio, Y. Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations. *J. Mach. Learn. Res.* **2017**, *18*, 187:1–187:30.
22. Courbariaux, M.; Bengio, Y. BinaryNet: Training Deep Neural Networks with Weights and Activations Constrained to +1 or -1. *CoRR* **2016**, *abs/1602.02830*.
23. Blott, M.; Preußer, T.B.; Fraser, N.J.; Gambardella, G.; O'Brien, K.; Umuroglu, Y.; Leeser, M.; Vissers, K.A. FINN-*R*: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks. *TRETS* **2018**, *11*, 16:1–16:23.
24. Rybalkin, V.; Pappalardo, A.; Ghaffar, M.M.; Gambardella, G.; Wehn, N.; Blott, M. FINN-L: Library Extensions and Design Trade-Off Analysis for Variable Precision LSTM Networks on FPGAs. FPL 2018, Dublin, Ireland, August 27-31, 2018. IEEE Computer Society, 2018, pp. 89–96.
25. Radu, V.; Kaszyk, K.; Wen, Y.; Turner, J.; Cano, J.; Crowley, E.J.; Franke, B.; Storkey, A.J.; O'Boyle, M.F.P. Performance Aware Convolutional Neural Network Channel Pruning for Embedded GPUs. IEEE International Symposium on Workload Characterization, IISWC 2019, Orlando, FL, USA, November 3-5, 2019, 2019, pp. 24–34.
26. Turner, J.; Crowley, E.J.; Radu, V.; Cano, J.; Storkey, A.; O'Boyle, M. Distilling with Performance Enhanced Students, 2019, [arXiv:stat.ML/1810.10460].
27. Dong, J.; Cheng, A.; Juan, D.; Wei, W.; Sun, M. DPP-Net: Device-Aware Progressive Search for Pareto-Optimal Neural Architectures. Computer Vision - ECCV 2018 - 15th European Conference, Munich, Germany, September 8-14, 2018, Proceedings, Part XI, 2018, pp. 540–555.
28. Guo, Y.; Yao, A.; Chen, Y. Dynamic Network Surgery for Efficient DNNs. Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain, 2016, pp. 1379–1387.
29. Stanley, K.O.; Miikkulainen, R. Evolving Neural Networks through Augmenting Topologies. *Evol. Comput.* **2002**, *10*, 99–127.
30. Huang, G.; Liu, S.; van der Maaten, L.; Weinberger, K.Q. CondenseNet: An Efficient DenseNet Using Learned Group Convolutions. 2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018. IEEE Computer Society, 2018, pp. 2752–2761.
31. Kim, Y.H.; Reddy, B.; Yun, S.; Seo, C. NEMO : Neuro-Evolution with Multiobjective Optimization of Deep Neural Network for Speed and Accuracy. AutoML Workshop, ICML, 2017.
32. Deb, K.; Agrawal, S.; Pratap, A.; Meyarivan, T. A fast and elitist multiobjective genetic algorithm: NSGA-II. *IEEE Trans. Evol. Comput.* **2002**, *6*, 182–197.
33. Intel. Intel Myriad X VPU Product Brief. https://www.intel.com/content/www/us/en/products/docs/processors/movidius-vpu/myriad-x-product-brief.html.
34. Su, J.; Fraser, N.J.; Gambardella, G.; Blott, M.; Durelli, G.; Thomas, D.B.; Leong, P.H.W.; Cheung, P.Y.K. Accuracy to Throughput Trade-Offs for Reduced Precision Neural Networks on Reconfigurable Logic. ARC 2018, Santorini, Greece, May 2-4, 2018, Proceedings, 2018, Lecture Notes in Computer Science, pp. 29–42.
35. Exploring knowledge distillation of Deep neural nets for efficient hardware solutions. CS230 Report http://cs230.stanford.edu/files_winter_2018/projects/6940224.pdf, 2018.

36. Hadka, D. Platypus: Multiobjective Optimization in Python. https://platypus.readthedocs.io, 2020.
37. Tian, Y.; Krishnan, D.; Isola, P. Contrastive Representation Distillation. International Conference on Learning Representations, 2020.
38. Qiu, J.; Wang, J.; Yao, S.; Guo, K.; Li, B.; Zhou, E.; Yu, J.; Tang, T.; Xu, N.; Song, S.; Wang, Y.; Yang, H. Going Deeper with Embedded FPGA Platform for Convolutional Neural Network. Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, Monterey, CA, USA, February 21-23, 2016. ACM, 2016, pp. 26–35.
39. Radu, V.; Kaszyk, K.; Wen, Y.; Turner, J.; Cano, J.; Crowley, E.J.; Franke, B.; Storkey, A.; O'Boyle, M. Performance Aware Convolutional Neural Network Channel Pruning for Embedded GPUs. IISWC 2019. IEEE, 2019.
40. Zhao, Y.; Gao, X.; Guo, X.; Liu, J.; Wang, E.; Mullins, R.; Cheung, P.Y.K.; Constantinides, G.A.; Xu, C. Automatic Generation of Multi-Precision Multi-Arithmetic CNN Accelerators for FPGAs. ICFPT 2019, Tianjin, China, December 9-13, 2019. IEEE, 2019, pp. 45–53.
41. Wang, J.; Lou, Q.; Zhang, X.; Zhu, C.; Lin, Y.; Chen, D. Design Flow of Accelerating Hybrid Extremely Low Bit-Width Neural Network in Embedded FPGA. FPL 2018, Dublin, Ireland, August 27-31. IEEE Computer Society, 2018, pp. 163–169.
42. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086.
43. Ding, C.; Wang, S.; Liu, N.; Xu, K.; Wang, Y.; Liang, Y. REQ-YOLO: A Resource-Aware, Efficient Quantization Framework for Object Detection on FPGAs. FPGA 2019, Seaside, CA, USA, February 24-26, 2019. ACM, 2019, pp. 33–42.
44. Gu, Q.; Ishii, I. Review of some advances and applications in real-time high-speed vision: Our views and experiences. *Int. J. Autom. Comput.* **2016**, *13*, 305–318.
45. Cheng, C.H. Towards Robust Direct Perception Networks for Automated Driving. IEEE Intelligent Vehicles Symposium (IV), Las Vegas, NV, USA. IEEE, 2020.
46. Xilinx. Brevitas: a Pytorch library for quantization-aware training. https://xilinx.github.io/brevitas.
47. Wang, H.; Xu, Y.; Ni, B.; Zhuang, L.; Xu, H. Flexible Network Binarization with Layer-Wise Priority. 2018 25th IEEE International Conference on Image Processing (ICIP), 2018, pp. 2346–2350.
48. Tian, Y.; Krishnan, D.; Isola, P. Contrastive Representation Distillation. 8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020. OpenReview.net, 2020.
49. Polino, A.; Pascanu, R.; Alistarh, D. Model compression via distillation and quantization. 6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings, 2018.
50. Cheng, Y.; Yu, F.X.; Feris, R.S.; Kumar, S.; Choudhary, A.N.; Chang, S. Fast Neural Networks with Circulant Projections. *CoRR* **2015**, *abs/1502.03436*.