LARQ COMPUTE ENGINE: DESIGN, BENCHMARK, AND DEPLOY STATE-OF-THE-ART BINARIZED NEURAL NETWORKS

Tom Bannink [*1] **Arash Bakthiari** [*2] **Adam Hillier** [*1] **Lukas Geiger** [*1] **Tim de Bruin** [1] **Leon Overweel** [1] **Jelmer Neeven** [1] **Koen Helwegen** [1]

ABSTRACT

We introduce Larq Compute Engine (LCE), a state-of-the-art Binarized Neural Network (BNN) inference engine, and use this framework to investigate several important questions about the efficiency of BNNs and to design a new leading BNN architecture. LCE provides highly optimized implementations of binary operations and accelerates binary convolutions by $8.5 - 18.5\times$ compared to their full-precision counterparts on Pixel 1 phones. LCE's integration with Larq and a sophisticated MLIR-based converter allow users to move smoothly from training to deployment. By extending TensorFlow and TensorFlow Lite, LCE supports models which combine binary and full-precision layers, and can be easily integrated into existing applications. Using LCE, we analyze the performance of existing BNN computer vision architectures and develop QuickNet, a simple, easy-to-reproduce BNN that outperforms existing binary networks in terms of latency and accuracy on ImageNet. Furthermore, we investigate the impact of full-precision shortcuts and the relationship between number of multiply-accumulate operations and model latency. We are convinced that empirical performance should drive BNN architecture design and hope this work will facilitate others to design, benchmark and deploy binary models.

1 INTRODUCTION AND MOTIVATION

There are many advantages in moving deep-learning-based computer vision computation from cloud datacenters to edge devices, including lower networking requirements, improved end-user privacy and real-time responses. In Binarized Neural Networks (BNNs) a significant proportion of weights and activations are restricted to the binary values, usually $\{-1, +1\}$. This considerably reduces model size and enables extremely efficient inference using `XOR` and `POPCOUNT` operations for binary multiplication and accumulation (Courbariaux & Bengio, 2016).

Although BNNs have the potential to make deep learning applications radically more efficient, in practice floating point or 8-bit quantized networks still dominate deep learning models in production. We see three key reasons for this. First, training BNNs is challenging due to the gradient mismatch problem and the need for optimizing discrete weights. Second, there is a lack of integrated software tooling that can be used to rapidly develop and deploy BNNs. Third, support for BNNs on existing hardware varies, and realizing the full potential of binarization requires custom hardware. In this work we focus on the second issue.

Deep learning inference frameworks like TensorFlow Lite (Google, 2017) have proven essential to the field of deep learning, both for research and for commercial development. These frameworks enable model development guided by direct performance measurements and quickly move from research to production. Although several BNN inference engines have been introduced, such as BMXNet (Yang et al., 2017), DaBNN (Zhang et al., 2019), and Riptide (Fromm et al., 2020), BNN research papers still tend to focus on operation counts and often lack empirical benchmarks.

In this paper, we introduce Larq Compute Engine (LCE), a state-of-the-art inference engine for Binarized Neural Networks. We built LCE with researcher ease-of-use as a top priority, and by integrating with the TensorFlow Keras (Abadi et al., 2015; Chollet, 2015) and Larq (Geiger & Team, 2020) ecosystems, we provide an end-to-end pipeline for training, benchmarking, and deploying BNNs. LCE includes a TensorFlow model graph converter and highly-optimized binarized custom operators for the TensorFlow Lite runtime (Google, 2017). LCE primarily targets 64-bit ARM devices, which includes all modern Android devices and the Raspberry Pi 3 and 4. We show that LCE is faster than existing inference engines for both individual convolutions and complete models, through benchmarks on the Pixel 1 Android phone and Raspberry Pi Model 4B board.

[*]Equal contribution [1]Plumerai Research. {tom, adamh, lukas, tim, leon, jelmer, koen}@plumerai.com [2]Work done while at Plumerai Research. a.bakhtiari@tum.de. Correspondence to: Koen Helwegen <koen@plumerai.com>.

Proceedings of the 4^{th} MLSys Conference, San Jose, CA, USA, 2021.

We also demonstrate the power of our integrated approach by providing real-world latency benchmarks for seven major BNNs from the literature and by introducing QuickNet, a new BNN model that uses a straightforward architecture and simple single-stage training method to achieve state-of-the-art performance. Finally, we use LCE to investigate several empirical questions about the design of BNN architectures: (a) What are major latency bottlenecks for BNNs from the literature? (b) What is the latency effect of different kinds of shortcut connections? and (c) How well do MACs correlate with real-world latency?

We hope that by introducing Larq Compute Engine we provide the BNN research community with a versatile tool for designing, benchmarking, and deploying binarized models. LCE is an actively-developed open-source project, available on GitHub at `larq/compute-engine`.

2 Background and Related Work

2.1 Efficient network design

While accuracy has been the primary goal of much deep learning computer vision research, model efficiency has been an increasingly important topic as DNNs have become larger and their usage has expanded. Particularly the potential for mobile and edge applications has given rise to an increasing focus on efficient network design over the past few years.

Initially, efforts were made to optimize network architectures for theoretical computational efficiency metrics. Iandola et al. (2016) used point-wise convolutions with squeeze and expand modules to reduce the number of network parameters. To reduce the number of operations in the network, MobileNets used depthwise separable convolutions (Howard et al., 2017), inverted residuals, and linear bottlenecks (Sandler et al., 2018). Grouped convolutions similarly helped reduce the number of operations with acceptable accuracy losses (Zhang et al., 2018; Huang et al., 2018).

More recently, state-of-the-art results have been obtained through architecture searches (Howard et al., 2019; Tan et al., 2019; Cai et al., 2020). Crucially, these searches optimize for the trade-off between accuracy and *measured* quantities such as inference time or energy usage.

Besides these efforts to develop more efficient architectures, there has been a lot of work on making existing networks more efficient through pruning (Han et al., 2015; 2016) and quantization (Zhu et al., 2017; Wang et al., 2019). In this paper we focus on the extreme case of the latter: Binarized Neural Networks (BNNs) with 1-bit weights and activations (Courbariaux & Bengio, 2016; Rastegari et al., 2016).

2.2 Binarized neural networks

Initial attempts of binarizing existing networks resulted in large accuracy drops on all but the most simple tasks (Rastegari et al., 2016; Phan et al., 2020). This motivated a host of more advanced training procedures for binarized networks (Courbariaux et al., 2015; Zhou et al., 2016; Peters & Welling, 2018; Alizadeh et al., 2019; Helwegen et al., 2019; Martínez et al., 2020; He et al., 2020), as well as changes to network architectures to make them more amenable to binarization and minimize accuracy degradation.

Some of these changes focused on more closely approximating higher bit-width networks. One approach has been to add full-precision scaling factors to the binarized weights in order to minimize the ℓ_2 distance from the corresponding full-precision weights (Rastegari et al., 2016; Bulat & Tzimiropoulos, 2019; Martínez et al., 2020); another has been to use a combination of multiple binarized branches to approximate individual weights (Lin et al., 2017) or network blocks (Zhuang et al., 2019). Zhu et al. (2019) used an ensemble of BNNs, reducing accuracy loss at the cost of increased binary computation. Additional residual shortcut connections for improved information flow have been found to be crucial for training more accurate BNN models (Liu et al., 2018; Bethge et al., 2019).

Recent BNN works have closed the gap with some of the popular higher bit-width architectures such as MobileNetv1 using custom network designs (Bethge et al., 2020) or by making these architectures more amenable to binarization through novel nonlinearities and training procedures (Liu et al., 2020). Neural Architecture Search has also recently been applied to BNNs (Shen et al., 2019; Bulat et al., 2020; Kim et al., 2020). However, while these recent works have made impressive gains in theoretical metrics, they lack evaluation on real hardware. This pursuit of higher accuracy can lead to network designs that might look good on paper, but are hard to implement efficiently (Fromm et al., 2020). We therefore argue to optimize for measured quantities such as inference time, and introduce the tools required to do so.

2.3 Existing frameworks for BNN inference

There are several existing solutions for BNN inference that use `XOR` and `POPCOUNT` operations to accelerate binary multiplication and accumulation. Here we provide a brief overview of the various frameworks and compare Larq Compute Engine to the most competitive solutions in Section 4.

BMXNet (Yang et al., 2017) is a BNN framework that extends MXNet (Chen et al., 2015), a general-purpose neural network training and inference library. BMXNet implements a 2D binarized convolution operation with im2col and a binary GEMM (GEneral Matrix Multiplication) kernel, written in C++, using `XOR` operators and

`__builtinpopcount` compiler intrinsics. By building on top of an existing framework BMXNet achieves broad model support without having to implement core full-precision neural network operators from scratch. For a large batch size of 200, on Intel x86-64 CPUs with a hardware popcount instruction, BMXNet claims a 13× speed-up for their 2D binarized convolution compared to a floating point implementation with ATLAS CBLAS (Whaley & Petitet, 2005). However, the C++ binary GEMM kernel compiles to machine code that is significantly slower than what can be achieved with optimised assembly kernels.

DaBNN (Zhang et al., 2019) is a stand-alone library for BNN inference on ARM devices. Like BMXNet, DaBNN implements 2D binarized convolutions with im2col and a binary GEMM kernel, written in hand-tuned 64-bit ARM assembly. DaBNN reports a $8 - 10\times$ speed-up for their 2D binarized convolution compared to a floating point implementation. It is very common for BNNs to include full-precision operators, such as in the first and last layers (Rastegari et al., 2016). As DaBNN doesn't extend an existing inference framework or runtime, all supported operators must be implemented from scratch and optimised for the target platform, which limits the space of model architectures that are supported. This approach maximises flexibility of implementation but significantly increases development cost and limits available features; for example, multi-threaded inference, depthwise floating point convolutions, image up- and down-sampling, as well as lower level mathematical operations like srqt, min, max or sigmoid are not supported.

TVM (Chen et al., 2018) is a compiler stack for deploying deep learning workloads on a diverse range of hardware back-ends. Instead of using hand-tuned optimised kernels for each operator on each target platform, the project aims to automatically generate fast kernels for running a specific model on a specific device. Built on TVM, **Riptide** (Fromm et al., 2020) is an end-to-end system for optimised BNN training and inference. Models are trained with TensorFlow, and the TVM TensorFlow graph converter is modified to add support for converting binarized operators such as 2D binarized convolutions. Riptide then extends TVM's code-generation to generate efficient kernels for these binarized operators targeting 32-bit ARM CPUs. Overall, Riptide reports a 4× to 12× speed-up of their BNN models compared to a floating-point implementation. A key focus is on reducing the overhead of intermediate 'glue' layers that commonly lie between pairs of binarized convolutions. Riptide replaces these layers (weight scaling, batch normalisation, and binary re-quantization) with a "fused binary glue operation" that, for example, replaces floating point multiplication by approximate scaling with a power-of-two integer shift. The fused binary glue an effective method, though it cannot be applied when there is a residual connection between the two binarized convolutions, as is common

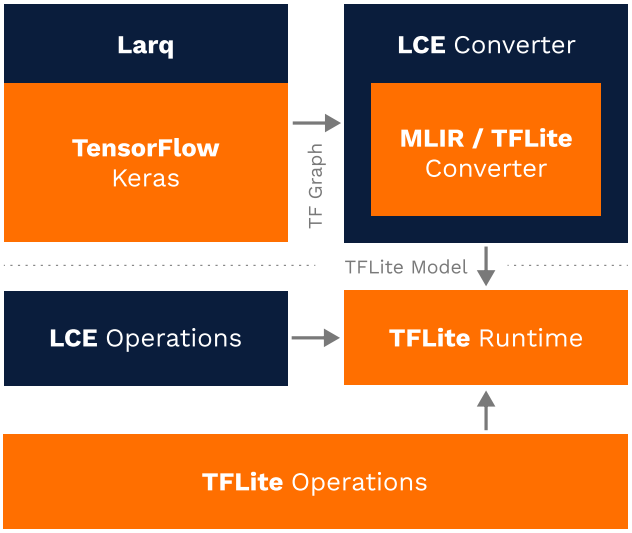


Figure 1. Larq Compute Engine workflow from training to deployment built on-top of the TensorFlow software stack (orange) from training (top), to deployment (bottom).

in recent BNN literature (Liu et al., 2018; Martínez et al., 2020; Bethge et al., 2020). The TVM compilation process means that Riptide has minimal runtime overhead, and will give good performance for a wide range of possible operators, but—as we show in Section 4—the generated kernels do not perform as well as hand-optimized assembly kernels.

3 Larq Compute Engine

We present Larq Compute Engine (LCE), an open-source BNN inference engine that outperforms existing BNN inference solutions. A high-level overview of the workflow from research to production is shown in Figure 1. LCE extends TensorFlow Lite (Google, 2017), which allows us to take advantage of the existing high-performance runtime and infrastructure for model conversion, benchmarking, and deployment. This makes it to easy adopt LCE in existing production applications. Together with BNN training library Larq (Geiger & Team, 2020) and the LCE converter, this forms an end-to-end solution for training, benchmarking and deploying BNNs. State-of-the-art models available in `Larq Zoo`, an open-source collection of BNNs, can be deployed directly with LCE.

3.1 Conversion to inference model

Usability is key to enable effective use of Larq Compute Engine for researchers exploring novel architecture designs. After building and training models with Larq (Geiger & Team, 2020), users must be able to easily convert their trained network to a TFLite model file that can be executed by our extended TFLite runtime. We achieve this by introducing a custom converter using the MLIR (Lattner et al., 2020) compiler infrastructure, allowing us to reuse most

of the existing TFLite conversion passes (Liu, 2019). This results in full support of all TFLite models and enables verification of the correctness of graph transformations applied during the conversion.

Larq, the Keras (Chollet, 2015)-based BNN training library, constructs a TensorFlow graph that emulates the BNN using floating point operations to approximate gradients during training. The main purpose of the converter is to transform this training graph into the TFLite model format for inference and replace the emulated binarized convolutions with truly binary, highly optimized LCE operations.

This infrastructure allows LCE to also handle weight scaling factors as used in Rastegari et al. (2016) and Liu et al. (2018), to fuse threshold-based activation functions as well as channel-wise multipliers and biases—commonly used by batch normalization (Ioffe & Szegedy, 2015)—into the preceding binarized convolution, and to support custom padding formats for faster inference as described in Section 3.2. These graph transformations are crucial for efficient inference as the overhead of full-precision channel-wise operations can become significant when full-precision convolutions are replaced with binary ones.

The MLIR compiler framework makes it straightforward to add more complicated graph optimisations, too. For example, if the output of one binarized convolution is passed through a threshold-based activation function and a batch normalization, and is then consumed by a second binarized convolution (without being used for a residual connection), there is no need to perform full-precision arithmetic or materialize the full-precision values at all. Instead, the direct accumulator output of the first convolution can be thresholded against pre-computed values to yield the binary input to the second convolution. The LCE converter performs these kinds of advanced optimizations automatically, without changes to training code or instruction from the user.

The final model conversion step, after the graph optimization passes, is binary weight compression. In the Larq model graph used for training, the binary weights are stored as float values, but in the LCE model file a single bit is used for each weight value, which reduces the size of the binary weights by a factor of 32.

The converter is available to download as part of the prebuilt `larq-compute-engine` PyPI package and exposes the model conversion functionality via a single API endpoint.

3.2 Operator implementations

Efficient binarized convolutions require not only a fast multiply-and-accumulation loop, but also careful design choices regarding padding, operator fusion, and more. In this section we will cover these topics and discuss some implementation details of the LCE operators.

LceQuantize

The binarized layers in LCE expect *bitpacked* input and are therefore preceded by an `LceQuantize` operator, which binarizes its input activations by extracting the sign bits[1]. Mathematically, a 0 valued bit represents a real value of 1.0 while 1 represents a real value of -1.0. For optimal memory access patterns, the number of channels is padded up to a multiple of 32[2]. At this point, the activation tensor is $32\times$ smaller than float input would be, and $8\times$ smaller than 8-bit quantized input. When the output of a binarized layer is directly used by another binarized layer (and is not required in higher precision, e.g. in a shortcut) then the first binarized layer can directly output bitpacked activations, eliminating this extra `LceQuantize` operation.

LceBConv2d

The primary binarized operator in LCE is a 2D binarized convolution, `LceBConv2d`. It accepts bitpacked input activations—for example, the output of a `LceQuantize` operator—and can write full-precision output or bitpacked output. The optimized implementation of `LceBConv2d` has three stages: first, a standard **im2col** procedure is used to rearrange the input activations in memory and reduce the convolution computation to a binary matrix multiplication; second, an optimized **BGEMM kernel** (Binary GEneral Matrix Multiplication) is used to perform the binary multiplication of the inputs with the weights and accumulate the results into 16-bit integers; and finally, an output-type-specific **output transformation** is applied which incorporates the fused channel-wise operators (see Section 3.1) and writes the final result to the output array.

For padded convolutions, which are very common, the im2col procedure fills the padded locations with zeros. As per the specification of the `LceQuantize` operator, these correspond to $+1.0$ values of the original input. We refer to this as one-padding to distinguish it from the default zero-padding in TensorFlow. Although LCE supports zero-padded binarized convolutions, this requires an extra correction step and is therefore slower. Larq provides the option to train binarized layers with one-padding; as we show in Section 5.1, using one-padding rather than zero-padding is not an impediment to training state-of-the-art BNNs.

The BGEMM kernel is implemented on top of the Ruy framework (Google, 2020), which is the GEMM library developed for use in TensorFlow Lite. This allows us to leverage optimization techniques available in Ruy such as tiling to maximize the number of cache hits, weight packing

[1] For completeness, LCE includes a `LceDequantize` operator which converts bitpacked data back into ± 1-valued float data.

[2] Common binarized networks already have multiples of 32 channels in all their binarized layers, so in practice no padding is performed.

Table 1. An analysis of the computational cost of performing float, 8-bit, and binary multiply and accumulate (MAC) operations using Neon SIMD instructions on the ARM Cortex-A76 CPU. In the float and 8-bit case, there exist specialized instructions that perform a fused multiplication and accumulation into 32-bit registers. Conversely, in the binary case no such fused instruction exists and so each step must be performed separately: `eor` for multiplication, `cnt` for 8-bit accumulation, and `addp` / `uadalp` for combining 8-bit results into 16-bit results. In the LCE BGEMM kernel we perform 1024 binary MACs using 24 instructions, which takes 13 cycles, or equivalently just over 78 MACs per cycle. Instruction throughput figures are sourced from the Cortex-A76 Software Optimization Guide (Arm Limited., 2019). Throughput figures are theoretical sustained maximums which assume optimal instruction scheduling without CPU pipeline stalls and do not account for potential latency from loading data into registers.

Precision	MAC instruction sequence	Throughput (instructions / cycle)	Throughput (MACs / cycle)
Float	`fmla`	2	8
8-bit	`sdot`	2	32
Binary	`eor`	2	78
	`cnt`	1	
	`addp` / `uadalp`	2 / 1	

to optimize memory access patterns, and multi-threading parallelization. At the core of the BGEMM kernel is 64-bit ARM assembly code that loads data (bitpacked weights and activations) from memory into CPU registers and performs the binary multiplication and accumulation operations (MACs). The code is optimized to maximize the use of the available vector register space so as to reduce weight-reloading from memory, and to reduce the frequency of CPU pipeline stalls according to the ARM Cortex-A Software Optimization Guide (Arm Limited., 2019). Loading binary data into registers is no different from loading float or 8-bit data; however, float and 8-bit MACs often benefit from dedicated CPU instructions which on current hardware platforms aren't available for binary MACs, which is why binarization speedups of 32× or 64× are unrealistic. Table 1 shows how float, 8-bit, and binary MACs can be implemented using Neon SIMD instructions and compares the theoretical maximum MAC throughputs. The speed of the binarized convolution also depends on how efficient the CPU cache can be used, which is where binarized layers have an advantage. For example, the weights of a binarized convolution with 256 filters of size 3×3 acting on 256 input channels take up 72 KiB of space which often fits entirely in the L2 cache, unlike the float or 8-bit equivalents. In Section 4 we present real world benchmarks of these implementations.

As discussed in Section 3.1, when writing full-precision output `LceBConv2d` supports a fused activation function and per-channel full-precision multiplier and bias. For full-precision convolutions, the fused multiplication can be performed "for free" because the multiplier values can be directly folded into the convolution weights and bias. For a binarized convolution with binary weights this is not an option, and so `LceBConv2d` has two extra inputs for per-channel full-precision values to be used as the multiplier and bias. These fused operations are performed directly on the BGEMM accumulator values, before they are written to memory, which avoids the extra read and write that would occur without operator fusing. Conversely, when writing bitpacked output the BGEMM accumulator values are compared with thresholds pre-computed in the converter to decide whether each output value is a one or zero bit.

LceBMaxPool2d

Networks that contain a full-precision MaxPool layer directly followed by a binarized convolution layer can be optimized by binarizing the activations *before* the MaxPool layer instead, since $\max(\text{sign}(X)) = \text{sign}(\max(X))$. The LCE converter recognizes this pattern automatically and emits the `LceBMaxPool2d` operator. It acts on data bitpacked by the `LceQuantize` operator and simply takes the bitwise `AND` to efficiently compute the binary maxpool.

4 BENCHMARKS

In this section we present various performance results measured using Larq Compute Engine. The measurements were taken on a Pixel 1 phone as well as a Raspberry Pi Model 4B with a 64-bit OS (Ubuntu LTS 20.04). The main text shows only the Pixel 1 benchmarks unless stated otherwise; the equivalent Raspberry Pi 4B numbers can be found in the appendix along with additional benchmarks on a Pixel 5 phone. All non-binary operators use the TensorFlow Lite implementation without modifications, and all TensorFlow Lite delegates are disabled.

4.1 The latency impact of binarizing convolutions

We first investigate the impact of binarization on individual convolutional layers. As an example we consider the four main convolutions that appear in ResNet18, a network architecture that has inspired numerous BNN designs. The

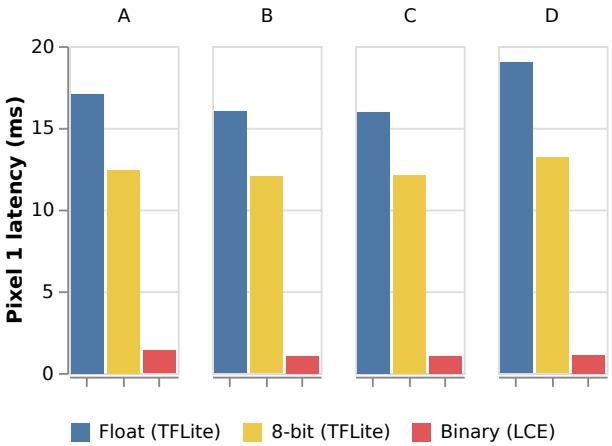


Figure 2. The impact of binarization on latency of convolutional layers with 3×3 kernels. We compare the latency of binarized convolutions to their equivalent 32-bit floating point or 8-bit integer versions for commonly used dimensions. In terms of height × width × in channels × out channels the convolutions are (A) 56×56×64×64; (B) 28×28×128×128; (C) 14×14×256×256; (D) 7×7×512×512. Compared to floating point, we observe binary speedups of between 12× for (A) and over 17× for (D). Compared to 8-bit, we observe speedups of between 9× and 12×.

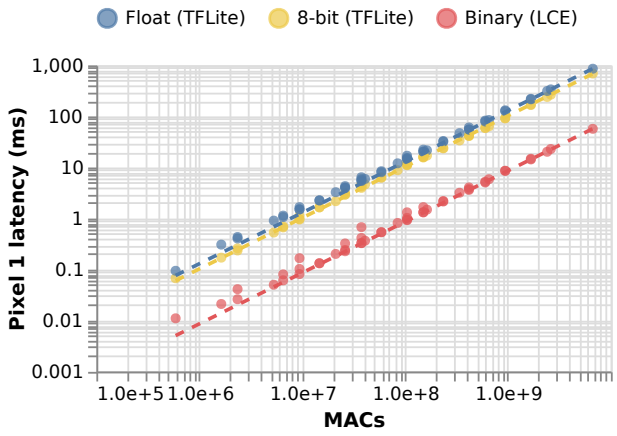


Figure 3. The relationship between MACs and latency for a large range of convolutions in binary, int8 and 32-bit floating point. Each convolution is a dot in the figure, while the dotted lines are least-square linear regressions between the MACs and latencies. Note that we use a log-log scale. We see an approximately linear relationship between MACs and latency in each precision, especially for larger dimensions. However, we also see substantial deviations from this linear relationship even for medium-sized convolutions. The input and output activations all have the same dimensions. Channels range from $\{32, 64, 96, 128, 160, 256\}$; input width and height range from $\{8, 16, 32, 64\}$ and kernel sizes are 3 or 5.

latencies of binarized versions of these are compared to their full-precision counterparts in Figure 2. We also benchmark 8-bit quantized versions of these convolutions, as near-lossless 8-bit quantization of networks like ResNet is now commonplace.

We see a large speedup across all four layers. Binarization reduces latency by $12 - 17\times$ compared to the floating-point implementation, with the largest performance gains being in the layers with the most channels.

In Section 3.2, we explained that on the ARMv8-A platform, the CPU instructions performing the binary MACs, allowed for a theoretical throughput of 78 binary MACs per clock cycle compared to 32 8-bit MACs or 8 float MACs, if we completely ignore memory reads and other operations. These theoretical numbers would suggest a 9.75× speedup over float and a 2.43× speedup over 8-bit convolutions. Memory reads, on the other hand, would be 32× and 8× faster, respectively. The actual speedup factors, as shown in Figure 2, turn out to be higher than these theoretical MAC throughput numbers. This can be attributed to memory reads and better cache efficiency for binarized layers.

Moving beyond a handful of examples, we next investigate a large space of convolutions of different dimensions. Channels range from $\{32, 64, 96, 128, 160, 256\}$; input width and height range from $\{8, 16, 32, 64\}$, and kernel sizes are 3×3 or 5×5. All included convolutions preserve the dimensions of the activation tensor, i.e. they have a stride of one, use equal padding and the number of input and output channels are the same. The number of MAC operations in the investigated blocks range from roughly 0.6 million to 6.5 million, and the floating point latency on a Pixel 1 ranges from 0.01 ms to over 850 ms.

The results are shown in Figure 3. We see that there is an approximately linear relationship between the number of MACs and latency for all three precisions. However, we also immediately see substantial deviations from this linear relationship. It is clear there is not a uniform speedup even when we constrain ourselves to 2D convolutions. These discrepancies can be caused by the overhead of bitpacking, im2col, and other operations which do not scale with MACs.

Although there is no universal speedup, we can give an approximate range of the efficiency gain one can expect from binarization on a Pixel 1. We can look at the speedup for each convolution benchmarked in Figure 3 individually and look at the range and the mean of these speedups. Arguably, speeding up larger convolutions is more important and so we also take a weighted mean, where speedups are weighted by the full-precision latency of the block. The results are summarized in Table 2.

It should be noted that this speedup is highly platform-dependent and may be very different on hardware platforms with different designs, instructions, or inference frameworks.

Table 2. Speedup of binarized convolutions on Pixel 1 with LCE, compared to 8-bit integer or floating point precision with TensorFlow Lite. We determine this speedup for a large range of individual convolutions and provide the mean, latency-weighted mean and overall range.

Precision	Mean	Weighted mean	Range
1 vs. 32	15.0×	15.1×	8.5–18.5×
1 vs. 8	10.8×	11.6×	6.1–13.4×

4.2 Comparison to other BNN inference frameworks

Figure 4 shows the latencies for the same binarized convolutions as in Figure 2 but now measured with the different inference frameworks introduced in Section 2.3. The kernel dimensions were chosen to match the sizes used in the BiRealNet architecture and the benchmarks presented in Zhang et al. (2019). These numbers were only measured on a Raspberry Pi 4B and not on a Pixel 1 phone because not all frameworks allowed deployment on the latter. Furthermore, Table 3 compares measurements of the overall latencies of BiRealNet, BinaryAlexNet and QuickNet (see Section 5.1) across different frameworks (Liu et al., 2018; Hubara et al., 2016). LCE significantly outperforms the other frameworks in the full model benchmarks and shows a clear performance improvement for single binarized convolutions for all kernel sizes except (A), where DaBNN is slightly faster due to its specialized implementation for this kernel size. LCE does not achieve the optimal performance in this setting since its im2col implementation performs optimally when height × width × in channels is a multiple of 128.

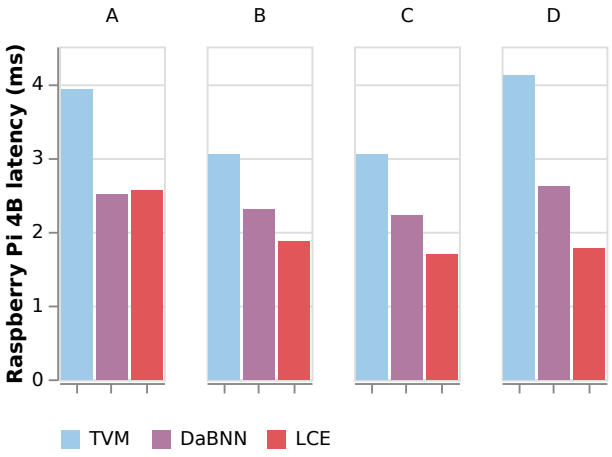


Figure 4. Comparison of the performance of LCE versus DaBNN and TVM on representative convolutions. The dimensions of the convolutions are the same as in Figure 2.

[3]The architecture was slightly modified compared to Hubara et al. (2016) to match the model used in Fromm et al. (2020).

Table 3. Comparison of full model latencies of LCE versus DaBNN and TVM measured in ms on a Raspberry Pi 4B. For a fair comparison, measurements excluding the first full precision block are shown in parentheses, as it dominates the inference time in some cases.

Library	BinaryAlexNet[3]	BiRealNet	QuickNet
DaBNN	unsupported	119.8 (58.8)	unsupported
TVM	647.8 (30.3)	862.9 (212.5)	245.5 (222.5)
LCE	**36.7** (13.7)	**87.0** (42.1)	**52.1** (46.0)

5 Designing BNNs using LCE

In this section we leverage LCE to design accurate and efficient BNNs and to analyse performance characteristics of commonly used binarized models in literature.

5.1 QuickNet: a simple, state-of-the-art BNN

The ability to evaluate on-device latency allows us to design novel architectures that are guaranteed to deliver on the desired accuracy-latency tradeoff. While this opens up many possibilities, including neural architecture search and large-scale hyperparameter tuning to discover novel architectures, in the following we strive for simplicity and aim to develop an efficient network architecture that is easy to train from scratch without the need for complex multi-stage training procedures and can serve as a baseline for future research.

Our architecture follows previous work (Liu et al., 2018; Martínez et al., 2020; Bethge et al., 2019) and uses four blocks $i \in 0, 1, 2, 3$, each consisting of N_i binary 3×3 convolutions with filter size k_i and residual connections over each layer. All binarized layers use one-padding (see Section 3.2) and ReLU activations (Glorot et al., 2011), and are followed by a batch normalization layer (Ioffe & Szegedy, 2015). Transition blocks between each residual section halve the spatial resolution and increase the filter count. After the final residual block, global average pooling and a full-precision fully connected layer are used to map to the 1000 classes used by ImageNet (Deng et al., 2009).

Using the detailed operation level profiling of LCE, we can analyse similar models and clearly identify bottlenecks in the network structure. The performance profiles of BinaryDenseNet28 and RealToBinaryNet (Bethge et al., 2019; Martínez et al., 2020) in Figure 5 clearly show the large impact of the first layer and other non-binary operations.

To improve the efficiency of the full-precision first layer while retaining competitive accuracy, we use a small 3×3 convolution with 16 filters and a depthwise separable convolution to increasing the feature size and decrease the spatial resolution from 224×224 to 56×56 using striding as shown in Figure 6a. The transition block (see Figure 6b) consists of a 3×3 antialiased max pooling (Zhang, 2019)—which

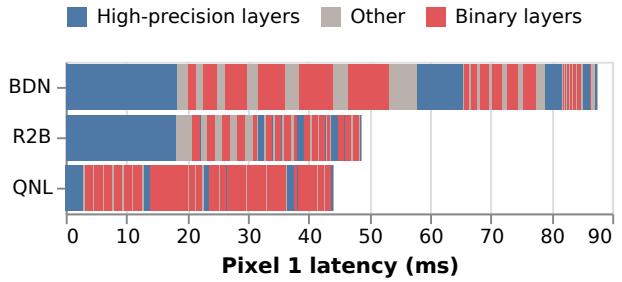
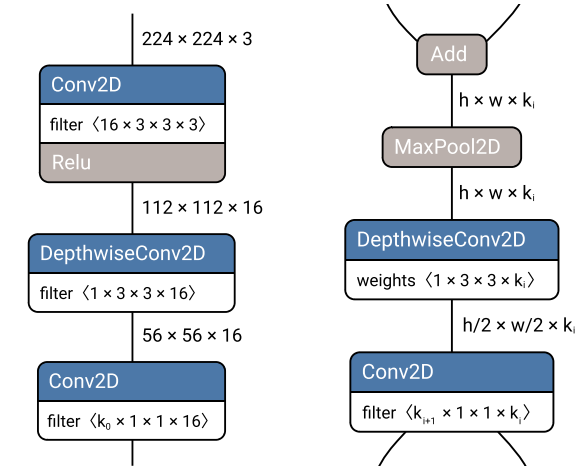


Figure 5. Breakdown of execution latencies stacked with respect to the layer number for three models: BinaryDensent28 (BDN), RealToBinaryNet (R2B) and QuickNet Large (QNL). This clearly shows the non-negligible runtime impact of non-binary operations in BinaryDenseNet and RealToBinaryNet as well as the significant impact of the first layer in those networks. QuickNet greatly improves in both of these areas resulting in a more efficient network.



(a) First convolutional block (b) Transition block

Figure 6. Full precision blocks in QuickNet, used for spatial downsampling of (a) the input and (b) the feature map of block i.

can be efficiently implemented by a max pooling layer and a strided depthwise convolution with a fixed blurring kernel—followed by a 1×1 full-precision convolutions with k_{i+1} filters to increase the feature size.

We train 3 models on the ImageNet dataset (Deng et al., 2009) for different latency targets and adjust the number of layers and filters according to Table 4. The networks are trained from scratch for 600 epochs on 4 NVIDIA V100 GPUs with a batch size of 2048 using the Adam optimizer (Kingma & Ba, 2015) with initial learning rate 0.01 and the straight-through estimator (Hubara et al., 2016) for binary weights and stochastic gradient descent with momentum 0.9 and learning rate of 0.1 for full-precision variables. We use a linear warmup over 5 epochs for both learning rates up to their initial value and decay to zero during training using a cosine schedule. Training images are preprocessed according to Tan & Le (2019) without AutoAugment (Cubuk et al., 2019) except for the largest model which slightly benefited from the additional augmentation. Table 4 lists the training and validation accuracies for all three models.

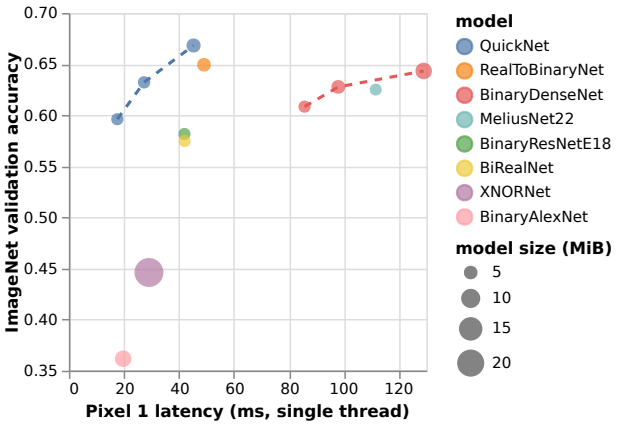


Figure 7. Latency and accuracy for various popular BNN model architectures on the ImageNet dataset.

Next we analyze the performance of QuickNet and compare against various popular binary architectures from the literature. Reference implementations and pretrained weights for all models discussed in this section are available in `larq zoo`. Although efficiency is a key motivation behind the development of binary architectures and algorithms, most of the original papers do not measure on-device latency and instead resort to indirect measurements such as number of MACs and binary operations.

Figure 7 shows the accuracies and latencies of QuickNet compared to various models from previous works (Liu et al., 2018; Hubara et al., 2016; Bethge et al., 2019; 2020; Martínez et al., 2020; Rastegari et al., 2016). We see that since the early AlexNet-based architectures, accuracies have improved substantially while memory footprints have markedly decreased. We can also observe that BiRealNet, RealToBinaryNet and QuickNet in particular have moved the pareto-front significantly forward, while other architectures such as BinaryDenseNet and MeliusNet have not fundamentally altered the landscape but rather trade higher accuracy against a worse latency.

Table 4. Number of layers per block $\boldsymbol{N}$ and number of filters $\boldsymbol{k}$ used in the QuickNet models and their top-1 ImageNet accuracies.

$\boldsymbol{N}$	$\boldsymbol{k}$	train (%)	eval (%)
(4, 4, 4, 4)	(32, 64, 256, 512)	59.9	59.4
(4, 4, 4, 4)	(64, 128, 256, 512)	64.3	63.3
(6, 8, 12, 6)	(64, 128, 256, 512)	59.1	66.9

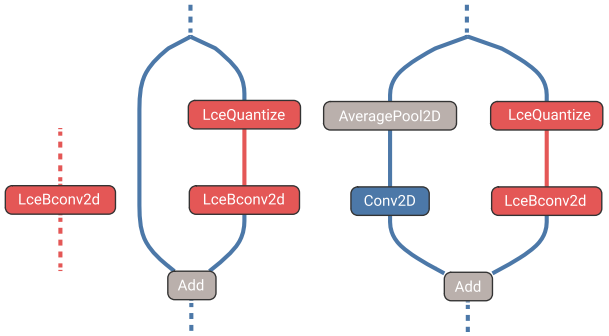


Figure 8. Overview of the different block types that are compared in Figure 9. Left: no shortcuts, input and output assumed to be binary. Middle: shortcut in a regular block. Right: shortcut in a downsampling layer, with a channel-doubling full-precision pointwise convolution in the shortcut.

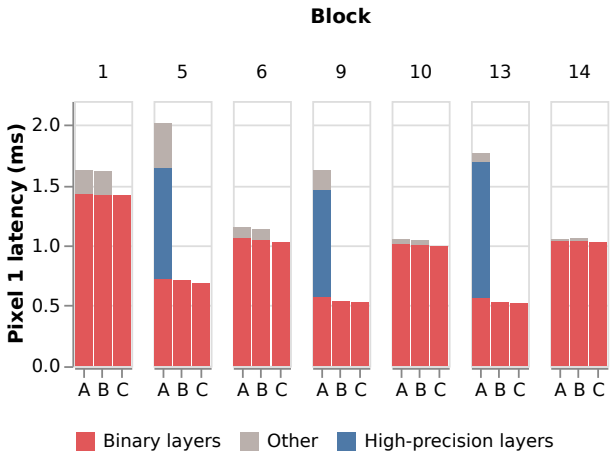


Figure 9. We compare three versions of a binarized ResNet18: (A) with shortcuts in every block; (B) with shortcuts in the regular blocks only; and (C) with no shortcuts anywhere. Repeated layers as well as the full-precision first and last layer are not shown. We see that the latency impact of shortcuts is small for regular blocks. Unsurprisingly, for downsampling blocks which contain an additional full-precision pointwise convolution, the cost of the additional pointwise convolution is substantial.

5.2 How do shortcuts affect BNN inference speed?

Since their introduction in Liu et al. (2018), full-precision shortcuts have been pervasive in BNN architectures due to the large improvement in accuracy they provide. Such shortcuts enable the preservation of full-precision information in the forward pass and may facilitate training by carrying non-distorted gradient signals during the backward pass. They are very attractive in theoretical metrics, as they do not increase memory footprint or the number of MACs.

However, they do impact the implementation of binarized networks. Whereas element-wise operations in a completely binarized architecture such as Binary AlexNet (Hubara et al., 2016) can be replaced with a single binarization function, full-precision shortcuts require normal evaluation of the transformations associated to batch normalization and activation functions. Existing work such as Fromm et al. (2020) emphasize the benefits of binarizing all intermediate activations. The question of their actual impact on latency is therefore of great practical significance.

To quantify the impact of full-precision shortcuts on latency, we perform latency measurements of different type of network blocks, as depicted in Figure 8. For the full-precision blocks, this only introduces an additional `Add` operation. For binarized operations, on the other hand, the shortcut introduces an `Add` but also forces the previous layer to write full-precision output rather than bitpacked data which means that the input activations of a subsequent binarized convolution need to be bitpacked separately, as indicated by the `LceQuantize` layer in the diagram. Nevertheless, as we can see in Figure 9, the speed-ups remain roughly equivalent to that of binarized convolutions without shortcuts, and their absolute impact of latency is small. Furthermore, Table 5 shows a breakdown of the contribution that each operator makes to overall latency of the QuickNet model, which makes it clear that the extra cost of binarized residual blocks comes from the full-precision `Add` rather than the more complex output transformation or extra bitpacking. These results suggest that at least on this type of hardware, the use of full-precision shortcuts really does drastically improve the pareto-front for BNNs.

5.3 Are MACs a useful proxy-metric for latency?

MACs are still commonly used to estimate the efficiency of models despite numerous warnings stating they are an unreliable guide when searching for efficient model designs (Wang et al., 2019; Tan et al., 2019; Ma et al., 2018). This question is even more complicated in the case of BNNs because in order to come up with a scalar metric it is necessary to assume a fixed relative performance between binarized and full-precision operations. The factor 64 or values close to it is often used in the literature, based on the theoretical argument that the complexity of multiplication grows with the square of the precision (Liu et al., 2018). Some papers

Table 5. Operation latency in QuickNet as a proportion of total latency, measured on a Raspberry Pi 4B. We split `LceBConv2d` into the main accumulation loop (binary multiplication and accumulation) and the output transformation (integer-to-float conversion, fused activation function, and fused batch normalization).

Operator	Latency (%)
`LceQuantize`	3.52
`LceBConv2d` (accumulation loop)	53.41
`LceBConv2d` (output transformation)	3.68
Full precision `Conv2D`	20.15
Full precision `Add`	9.55
All other full precision	9.69

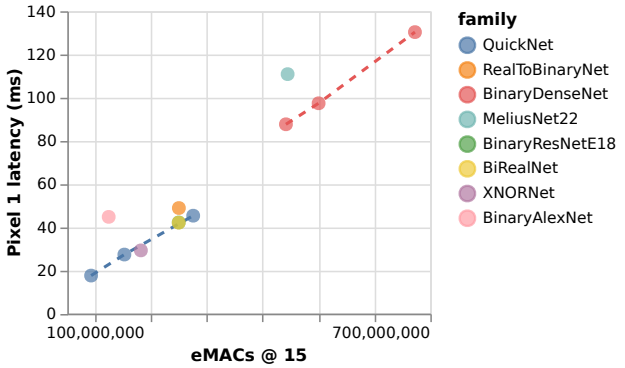


Figure 10. The relationship between MACs and latency for the BNNs in `Larq Zoo`. Here we assume a scaling of 15 binary MACs per full-precision MAC—the combined number is referred to as eMACs to indicate the assumed equivalence. We see MACs as a useful metric for comparing models with a similar design, but not when comparing entirely different architectures.

use the factor 58 (Zhu et al., 2019; Munagala et al., 2020) based on the results in Rastegari et al. (2016), but this work provides no details about absolute latencies or whether any optimizations where used in the baseline benchmarks.

To get better insight into the usefullness of MACs in estimating model performance, we compare MACs and latency for the models in `Larq Zoo`. Based on the results in section 4.1, we assume 15 binary MACs are equivalent to one floating point MAC. The results are shown in Figure 10. We see that within models of the same family (e.g. QuickNets, BinaryDenseNets) MACs can be a reasonable proxy for latency. However, when comparing different model designs the relationship breaks down. For example, BinaryAlexNet is almost $2\times$ slower than models with the same number of MACs, while matching the latency of models with over $3\times$ the number of MACs. These observations confirm that MACs have limited value when exploring new types of model designs, and cannot substitute empirical benchmarks of latency or other key performance metrics.

6 Conclusion

This paper introduces Larq Compute Engine, a state-of-the-art inference engine for Binarized Neural Networks. Built on top of TensorFlow and TensorFlow Lite, LCE and Larq provide an end-to-end solution for training BNNs and benchmarking them on mobile devices. The highly optimized BGEMM kernels in LCE provide speedups of $8.5\times$ to $18.5\times$ on Pixel 1 phones, while a MLIR-based converter handles the mapping from training graph to inference model, taking care of converting the emulated binary operations used during training to true binarized operations and the management of bitpacking and activation precision throughout the network. This brings the software infrastructure for deploying BNNs to the same level as the infrastructure for higher precision models provided by TensorFlow and TensorFlow Lite, thus resolving one of the key obstacles to wide-scale usage of BNNs.

With LCE in place, we have been able to investigate several questions with practical importance to the development of BNNs. First, we have identified latency bottlenecks in existing network designs and shown that full-precision parts of architectures are often a major component of the overall latency of the models. Using these insights we have been able to design QuickNet—a simple, easy to reproduce binary architecture that outperforms existing binarized model in terms of accuracy and latency while using a larger fraction of binarized operations. Second, we have looked in more detail at the impact of full-precision shortcuts on latency, a topic of some controversy in the literature. We have demonstrated that although shortcuts bring some overhead in terms of latency the additional overhead is marginal and well worth the accuracy gains provided by these shortcuts. Third, we have investigated the value of MACs in designing new architectures and we have confirmed the number of MAC is a poor predictor for latency when comparing highly divergent architecture designs.

We are very excited about the future of binarized models—we see numerous opportunities and hope that LCE will facilitate further progress in the field. While QuickNet is a state-of-the-art binarized model, it is only a first step in the measurement-driven design of neural networks. On the architecture side, it has now become possible to unify the emerging field of binarized neural architecture search with the hardware-in-the-loop based approaches that have generated so much progress for full-precision models. We also note we have not focused on training methods here, and we expect QuickNet can improve further by applying more sophisticated methods such as knowledge distillation. Above all, we hope that by providing a software framework that is high quality, easy to use and fully integrated, we will lower the barrier to experimentation with entirely novel designs and algorithms.

Finally, we want to note that hardware is a crucial component in the road towards efficient deep learning, and deployment to 64-bit ARM devices such as mobile phones is only a first step for bringing BNNs to the real world. As we discussed, most existing hardware platforms come with specialized support for full-precision or 8-bit matrix multiplication, such as vectorized MAC instructions, without providing the binarized counterpart of such operations. There is an opportunity for further large performance improvements through customized hardware for BNNs.

Acknowledgements

We are grateful to all contributors to Larq Compute Engine and the Larq Ecosystem.

REFERENCES

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., and Zheng, X. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL https://www.tensorflow.org/. Software available from tensorflow.org.

Alizadeh, M., Fernández-Marqués, J., Lane, N. D., and Gal, Y. An empirical study of binary neural networks' optimisation. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL https://openreview.net/forum?id=rJfUCoR5KX.

Arm Limited. Arm Cortex-A76 Software Optimization Guide, 2019. URL https://developer.arm.com/documentation/swog307215/a/.

Bethge, J., Yang, H., Bornstein, M., and Meinel, C. Back to simplicity: How to train accurate bnns from scratch? *CoRR*, abs/1906.08637, 2019. URL https://arxiv.org/abs/1906.08637.

Bethge, J., Bartz, C., Yang, H., Chen, Y., and Meinel, C. Meliusnet: Can binary neural networks achieve mobilenet-level accuracy? *CoRR*, abs/2001.05936, 2020. URL https://arxiv.org/abs/2001.05936.

Bulat, A. and Tzimiropoulos, G. Xnor-net++: Improved binary neural networks. In *30th British Machine Vision Conference 2019, BMVC 2019, Cardiff, UK, September 9-12, 2019*, pp. 62. BMVA Press, 2019. URL https://bmvc2019.org/wp-content/uploads/papers/0121-paper.pdf.

Bulat, A., Martínez, B., and Tzimiropoulos, G. BATS: binary architecture search. *CoRR*, abs/2003.01711, 2020. URL https://arxiv.org/abs/2003.01711.

Cai, H., Gan, C., Wang, T., Zhang, Z., and Han, S. Once-for-all: Train one network and specialize it for efficient deployment. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=HylxE1HKwS.

Chen, T., Li, M., Li, Y., Lin, M., Wang, N., Wang, M., Xiao, T., Xu, B., Zhang, C., and Zhang, Z. Mxnet: A flexible and efficient machine learning library for heterogeneous distributed systems. *CoRR*, abs/1512.01274, 2015. URL http://arxiv.org/abs/1512.01274.

Chen, T., Moreau, T., Jiang, Z., Zheng, L., Yan, E. Q., Shen, H., Cowan, M., Wang, L., Hu, Y., Ceze, L., Guestrin, C., and Krishnamurthy, A. TVM: an automated end-to-end optimizing compiler for deep learning. In Arpaci-Dusseau, A. C. and Voelker, G. (eds.), *13th USENIX Symposium on Operating Systems Design and Implementation, OSDI 2018, Carlsbad, CA, USA, October 8-10, 2018*, pp. 578–594. USENIX Association, 2018. URL https://www.usenix.org/conference/osdi18/presentation/chen.

Chollet, F. Keras. https://keras.io, 2015.

Courbariaux, M. and Bengio, Y. Binarynet: Training deep neural networks with weights and activations constrained to +1 or -1. *CoRR*, abs/1602.02830, 2016. URL http://arxiv.org/abs/1602.02830.

Courbariaux, M., Bengio, Y., and David, J. Binaryconnect: Training deep neural networks with binary weights during propagations. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 3123–3131, 2015. URL http://papers.nips.cc/paper/5647-binaryconnect-training-deep-neural-networks-with-binary-weights-during-propagations.

Cubuk, E. D., Zoph, B., Mane, D., Vasudevan, V., and Le, Q. V. Autoaugment: Learning augmentation strategies from data. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 113–123. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00020. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Cubuk_AutoAugment_Learning_Augmentation_Strategies_From_Data_CVPR_2019_paper.html.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., and Li, F. Imagenet: A large-scale hierarchical image database. In *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2009), 20-25 June 2009, Miami, Florida, USA*, pp. 248–255. IEEE Computer Society, 2009. doi: 10.1109/CVPR.2009.5206848. URL https://doi.org/10.1109/CVPR.2009.5206848.

Fromm, J., Cowan, M., Philipose, M., Ceze, L., and Patel, S. N. Riptide: Fast end-to-end binarized neural networks. In Dhillon, I. S., Papailiopoulos, D. S., and Sze, V. (eds.),

Proceedings of Machine Learning and Systems 2020, MLSys 2020, Austin, TX, USA, March 2-4, 2020. mlsys.org, 2020. URL https://proceedings.mlsys.org/book/312.pdf.

Geiger, L. and Team, P. Larq: An open-source library for training binarized neural networks. *Journal of Open Source Software*, 5(45):1746, 2020. doi: 10.21105/joss.01746. URL https://doi.org/10.21105/joss.01746.

Glorot, X., Bordes, A., and Bengio, Y. Deep sparse rectifier neural networks. In Gordon, G. J., Dunson, D. B., and Dudík, M. (eds.), *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics, AISTATS 2011, Fort Lauderdale, USA, April 11-13, 2011*, volume 15 of *JMLR Proceedings*, pp. 315–323. JMLR.org, 2011. URL http://proceedings.mlr.press/v15/glorot11a/glorot11a.pdf.

Google. TensorFlow Lite, 2017. URL https://www.tensorflow.org/lite.

Google. *The RUY matrix multiplication library*, 2020. URL https://github.com/google/ruy/.

Han, S., Pool, J., Tran, J., and Dally, W. J. Learning both weights and connections for efficient neural network. In Cortes, C., Lawrence, N. D., Lee, D. D., Sugiyama, M., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pp. 1135–1143, 2015. URL http://papers.nips.cc/paper/5784-learning-both-weights-and-connections-for-efficient-neural-network.

Han, S., Mao, H., and Dally, W. J. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. In Bengio, Y. and LeCun, Y. (eds.), *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*, 2016. URL http://arxiv.org/abs/1510.00149.

He, X., Mo, Z., Cheng, K., Xu, W., Hu, Q., Wang, P., Liu, Q., and Cheng, J. ProxyBNN: Learning binarized neural networks via proxy matrices. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part III*, volume 12348 of *Lecture Notes in Computer Science*, pp. 223–241. Springer, 2020. doi: 10.1007/978-3-030-58580-8_14. URL https://doi.org/10.1007/978-3-030-58580-8_14.

Helwegen, K., Widdicombe, J., Geiger, L., Liu, Z., Cheng, K., and Nusselder, R. Latent weights do not exist: Rethinking binarized neural network optimization. In Wallach, H. M., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E. B., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pp. 7531–7542, 2019. URL http://papers.nips.cc/paper/8971-latent-weights-do-not-exist-rethinking-binarized-neural-network-optimization.

Howard, A., Pang, R., Adam, H., Le, Q. V., Sandler, M., Chen, B., Wang, W., Chen, L., Tan, M., Chu, G., Vasudevan, V., and Zhu, Y. Searching for mobilenetv3. In *2019 IEEE/CVF International Conference on Computer Vision, ICCV 2019, Seoul, Korea (South), October 27 - November 2, 2019*, pp. 1314–1324. IEEE, 2019. doi: 10.1109/ICCV.2019.00140. URL https://doi.org/10.1109/ICCV.2019.00140.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., Andreetto, M., and Adam, H. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *CoRR*, abs/1704.04861, 2017. URL http://arxiv.org/abs/1704.04861.

Huang, G., Liu, S., van der Maaten, L., and Weinberger, K. Q. Condensenet: An efficient densenet using learned group convolutions. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 2752–2761. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00291. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Huang_CondenseNet_An_Efficient_CVPR_2018_paper.html.

Hubara, I., Courbariaux, M., Soudry, D., El-Yaniv, R., and Bengio, Y. Binarized neural networks. In Lee, D. D., Sugiyama, M., von Luxburg, U., Guyon, I., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pp. 4107–4115, 2016. URL http://papers.nips.cc/paper/6573-binarized-neural-networks.

Iandola, F. N., Moskewicz, M. W., Ashraf, K., Han, S., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size. *CoRR*, abs/1602.07360, 2016. URL http://arxiv.org/abs/1602.07360.

Ioffe, S. and Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Bach, F. R. and Blei, D. M. (eds.), *Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6-11 July 2015*, volume 37 of *JMLR Workshop and Conference Proceedings*, pp. 448–456. JMLR.org, 2015. URL http://proceedings.mlr.press/v37/ioffe15.html.

Kim, D., Singh, K. P., and Choi, J. Learning architectures for binary networks. In Vedaldi, A., Bischof, H., Brox, T., and Frahm, J. (eds.), *Computer Vision - ECCV 2020 - 16th European Conference, Glasgow, UK, August 23-28, 2020, Proceedings, Part XII*, volume 12357 of *Lecture Notes in Computer Science*, pp. 575–591. Springer, 2020. doi: 10.1007/978-3-030-58610-2_34. URL https://doi.org/10.1007/978-3-030-58610-2_34.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y. (eds.), *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, 2015. URL http://arxiv.org/abs/1412.6980.

Lattner, C., Pienaar, J. A., Amini, M., Bondhugula, U., Riddle, R., Cohen, A., Shpeisman, T., Davis, A., Vasilache, N., and Zinenko, O. MLIR: A compiler infrastructure for the end of Moore's law. *CoRR*, abs/2002.11054, 2020. URL https://arxiv.org/abs/2002.11054.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In Guyon, I., von Luxburg, U., Bengio, S., Wallach, H. M., Fergus, R., Vishwanathan, S. V. N., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pp. 345–353, 2017. URL http://papers.nips.cc/paper/6638-towards-accurate-binary-convolutional-neural-network.

Liu, F. Building TensorFlow converter tools with MLIR, 2019. URL https://drive.google.com/file/d/1ZCLTiEm5cVON34JrnUTvm5XdhjYz3DXV.

Liu, Z., Wu, B., Luo, W., Yang, X., Liu, W., and Cheng, K. Bi-real net: Enhancing the performance of 1-bit cnns with improved representational capability and advanced training algorithm. In Ferrari, V., Hebert, M., Sminchisescu, C., and Weiss, Y. (eds.), *Computer Vision – ECCV 2018*, pp. 747–763, Cham, 2018. Springer International Publishing. ISBN 978-3-030-01267-0. doi: 10.1007/978-3-030-01267-0_44.

Liu, Z., Shen, Z., Savvides, M., and Cheng, K. Reactnet: Towards precise binary neural network with generalized activation functions. *CoRR*, abs/2003.03488, 2020. URL https://arxiv.org/abs/2003.03488.

Ma, N., Zhang, X., Zheng, H. T., and Sun, J. Shufflenet V2: Practical guidelines for efficient cnn architecture design. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11218 LNCS:122–138, 2018. ISSN 16113349. doi: 10.1007/978-3-030-01264-9_8.

Martínez, B., Yang, J., Bulat, A., and Tzimiropoulos, G. Training binary neural networks with real-to-binary convolutions. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020. URL https://openreview.net/forum?id=BJg4NgBKvH.

Munagala, S. A., Prabhu, A., and Namboodiri, A. M. Stq-nets: Unifying network binarization and structured pruning. In *31st British Machine Vision Conference 2020, BMVC 2020, Virtual Event, UK, September 7-10, 2020*. BMVA Press, 2020. URL https://www.bmvc2020-conference.com/assets/papers/0113.pdf.

Peters, J. W. T. and Welling, M. Probabilistic binary neural networks. *CoRR*, abs/1809.03368, 2018. URL http://arxiv.org/abs/1809.03368.

Phan, H., Huynh, D., He, Y., Savvides, M., and Shen, Z. Mobinet: A mobile binary network for image classification. In *IEEE Winter Conference on Applications of Computer Vision, WACV 2020, Snowmass Village, CO, USA, March 1-5, 2020*, pp. 3442–3451. IEEE, 2020. doi: 10.1109/WACV45572.2020.9093444. URL https://doi.org/10.1109/WACV45572.2020.9093444.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. In Leibe, B., Matas, J., Sebe, N., and Welling, M. (eds.), *Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV*, volume 9908 of *Lecture Notes in Computer Science*, pp. 525–542. Springer, 2016. doi: 10.1007/978-3-319-46493-0_32. URL https://doi.org/10.1007/978-3-319-46493-0_32.

Sandler, M., Howard, A. G., Zhu, M., Zhmoginov, A., and Chen, L. Mobilenetv2: Inverted residuals and linear bottlenecks. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 4510–4520. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00474. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Sandler_MobileNetV2_

Inverted_Residuals_CVPR_2018_paper.html.

Shen, M., Han, K., Xu, C., and Wang, Y. Searching for accurate binary neural architectures. In *2019 IEEE/CVF International Conference on Computer Vision Workshops, ICCV Workshops 2019, Seoul, Korea (South), October 27-28, 2019*, pp. 2041–2044. IEEE, 2019. doi: 10.1109/ICCVW.2019.00256. URL https://doi.org/10.1109/ICCVW.2019.00256.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 6105–6114. PMLR, 2019. URL http://proceedings.mlr.press/v97/tan19a.html.

Tan, M., Chen, B., Pang, R., Vasudevan, V., Sandler, M., Howard, A., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 2820–2828. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00293. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Tan_MnasNet_Platform-Aware_Neural_Architecture_Search_for_Mobile_CVPR_2019_paper.html.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. HAQ: hardware-aware automated quantization with mixed precision. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2019, Long Beach, CA, USA, June 16-20, 2019*, pp. 8612–8620. Computer Vision Foundation / IEEE, 2019. doi: 10.1109/CVPR.2019.00881. URL http://openaccess.thecvf.com/content_CVPR_2019/html/Wang_HAQ_Hardware-Aware_Automated_Quantization_With_Mixed_Precision_CVPR_2019_paper.html.

Whaley, R. C. and Petitet, A. Minimizing development and maintenance costs in supporting persistently optimized BLAS. *Software: Practice and Experience*, 35(2):101–121, February 2005.

Yang, H., Fritzsche, M., Bartz, C., and Meinel, C. Bmxnet: An open-source binary neural network implementation based on mxnet. In Liu, Q., Lienhart, R., Wang, H., Chen, S. K., Boll, S., Chen, Y. P., Friedland, G., Li, J., and Yan, S. (eds.), *Proceedings of the 2017 ACM on Multimedia Conference, MM 2017, Mountain View, CA, USA, October 23-27, 2017*, pp. 1209–1212. ACM, 2017. doi: 10.1145/3123266.3129393. URL https://doi.org/10.1145/3123266.3129393.

Zhang, J., Pan, Y., Yao, T., Zhao, H., and Mei, T. dabnn: A super fast inference framework for binary neural networks on ARM devices. In Amsaleg, L., Huet, B., Larson, M. A., Gravier, G., Hung, H., Ngo, C., and Ooi, W. T. (eds.), *Proceedings of the 27th ACM International Conference on Multimedia, MM 2019, Nice, France, October 21-25, 2019*, pp. 2272–2275. ACM, 2019. doi: 10.1145/3343031.3350534. URL https://doi.org/10.1145/3343031.3350534.

Zhang, R. Making convolutional networks shift-invariant again. In Chaudhuri, K. and Salakhutdinov, R. (eds.), *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pp. 7324–7334. PMLR, 2019. URL http://proceedings.mlr.press/v97/zhang19a.html.

Zhang, X., Zhou, X., Lin, M., and Sun, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. In *2018 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2018, Salt Lake City, UT, USA, June 18-22, 2018*, pp. 6848–6856. IEEE Computer Society, 2018. doi: 10.1109/CVPR.2018.00716. URL http://openaccess.thecvf.com/content_cvpr_2018/html/Zhang_ShuffleNet_An_Extremely_CVPR_2018_paper.html.

Zhou, S., Wu, Y., Ni, Z., Zhou, X., Wen, H., and Zou, Y. DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients. *arXiv preprint arXiv:1606.06160*, 2016. doi: 10.1080/00131940802117563. URL http://arxiv.org/abs/1606.06160.

Zhu, C., Han, S., Mao, H., and Dally, W. J. Trained ternary quantization. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017. URL https://openreview.net/forum?id=S1_pAu9xl.

Zhu, S., Dong, X., and Su, H. Binary ensemble neural network: More bits per network or more networks per bit? In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4918–4927, June 2019. doi: 10.1109/CVPR.2019.00506. URL http://openaccess.thecvf.com/content_CVPR_2019/papers/Zhu_Binary_Ensemble_Neural_Network_More_Bits_per_Network_or_More_CVPR_2019_paper.pdf.

Zhuang, B., Shen, C., Tan, M., Liu, L., and Reid, I. Structured binary neural networks for accurate image classification and semantic segmentation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 413–422, June 2019. doi: 10.1109/CVPR.2019.00050. URL http://openaccess.thecvf.com/content_CVPR_2019/papers/Zhuang_Structured_Binary_Neural_Networks_for_Accurate_Image_Classification_and_Semantic_CVPR_2019_paper.pdf.

A BENCHMARKS ON RASPBERRY PI 4B

We provide the benchmark numbers on the Raspberry Pi 4B, for comparison with the numbers on a Pixel 1 phone presented in the main text.

For the example convolutions, the performance for the various precisions is shown in Figure 11. The relationship between MACs and latency of individual convolutions is shown in Figure 12. The associated speedups are described in Table 6. We see that speedups with respect to floating point convolutions is slightly better while improvement with respect to 8-bit quantized convolutions is a bit lower.

The accuracy and latency of existing BNN models and QuickNet on the Raspberry Pi is shown in Figure 13. The latency for various shortcut configurations are compared in Figure 14, and the relationship between MACs and latency is shown in Figure 15.

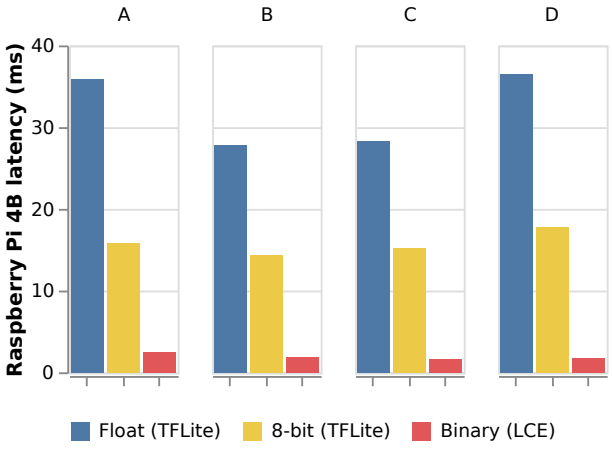


Figure 11. The impact of binarization on latency of convolutional layers with 3×3 kernel for the same convolutions as in Figure 2 on a Raspberry Pi 4B. With respect to floating point, we observe binary speedups of between 14× for (A) and over 20× for (D). With respect to 8-bit, we observe speedups of between 6× and 10×.

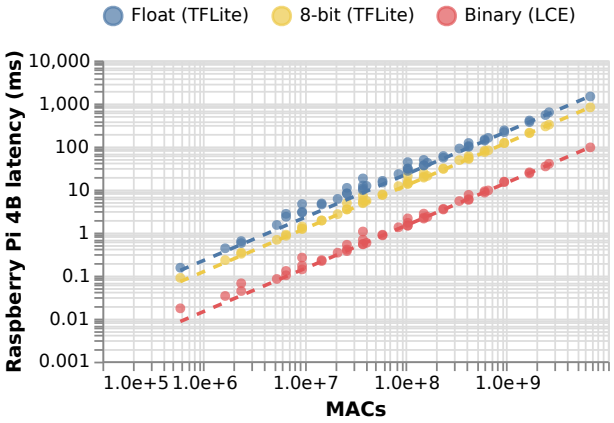


Figure 12. The relationship between MACs and latency for a large range of convolutions in binary, int8 and 32-bit floating point. See also Figure 3.

Table 6. Speedup of binarized convolutions on Raspberry Pi 4B with LCE, compared to 8-bit integer or floating point precision with TensorFlow Lite. We determine this speedup for a large range of individual convolutions and provide the mean, latency-weighted mean and overall range. Compare to Table 2.

Precision	Mean	Weighted mean	Range
1 vs. 32	17.5×	16.0×	8.8–23.0×
1 vs. 8	8.3×	8.5×	5.1–9.6×

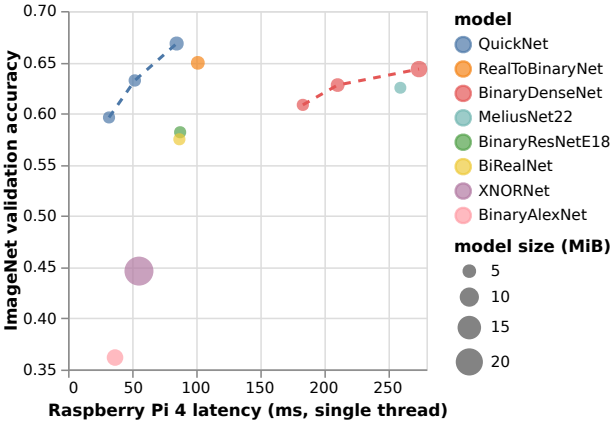


Figure 13. Accuracy and latency for various popular BNN models as well as the newly introduced QuickNet on a Raspberry Pi 4B. Compare to Figure 7.

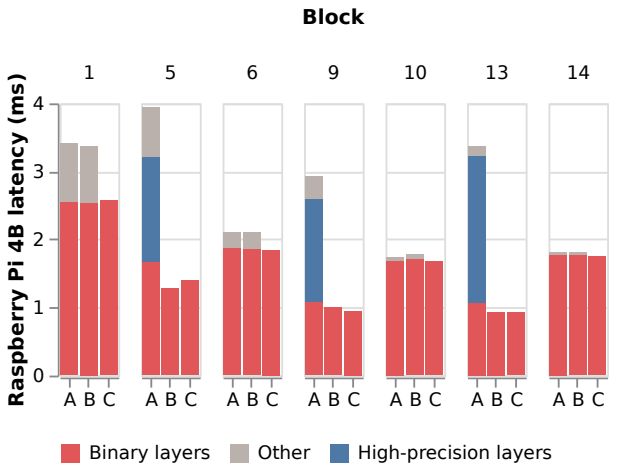


Figure 14. Study of the impact of full-precision shortcuts on latency on a Raspberry Pi 4B. See Figure 9 for details.

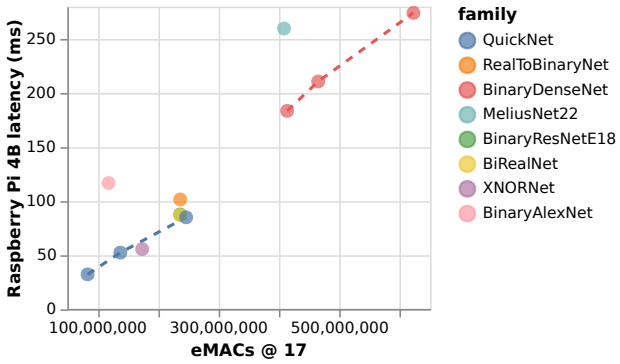


Figure 15. The relationship between MACs and latency for the BNNs in `Larq Zoo`. Based on Table 6, here we assume a scaling of 17 binary MACs per full-precision MAC - the combined number is referred to as eMACs to indicate the assumed equivalence. Compare to Figure 10.

B BENCHMARKS ON PIXEL 5 PHONE

We provide additional benchmark numbers on the Pixel 5 phone running Android 11 in Figure 16. The Cortex-A76 CPU used in the Pixel 5 phone includes the `sdot` instruction which significantly improves performance of 8-bit convolutions compared to the Pixel 1 and Raspberry PI 4B.

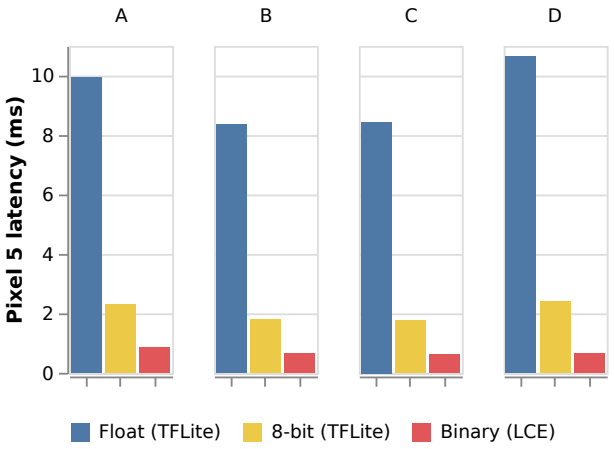


Figure 16. The impact of binarization on latency of convolutional layers with 3×3 kernel for the same convolutions as in Figure 2 on a Pixel 5 phone. With respect to floating point, we observe binary speedups of between 11.5× for (A) and 16.5× for (D). With respect to 8-bit, we observe speedups of between 2.7× and 3.7×.