SEARCHING FOR WINOGRAD-AWARE QUANTIZED NETWORKS

Javier Fernandez-Marques [1] **Paul N. Whatmough** [2] **Andrew Mundy** [2] **Matthew Mattina** [2]

ABSTRACT

Lightweight architectural designs of Convolutional Neural Networks (CNNs) together with quantization have paved the way for the deployment of demanding computer vision applications on mobile devices. Parallel to this, alternative formulations to the convolution operation such as FFT, Strassen and Winograd, have been adapted for use in CNNs offering further speedups. Winograd convolutions are the fastest known algorithm for spatially small convolutions, but exploiting their full potential comes with the burden of numerical error, rendering them unusable in quantized contexts. In this work we propose a Winograd-aware formulation of convolution layers which exposes the numerical inaccuracies introduced by the Winograd transformations to the learning of the model parameters, enabling the design of competitive quantized models without impacting model size. We also address the source of the numerical error and propose a relaxation on the form of the transformation matrices, resulting in up to 10% higher classification accuracy on CIFAR-10. Finally, we propose wiNAS, a neural architecture search (NAS) framework that jointly optimizes a given macro-architecture for accuracy and latency leveraging Winograd-aware layers. A Winograd-aware ResNet-18 optimized with wiNAS for CIFAR-10 results in $2.66\times$ speedup compared to *im2row*, one of the most widely used optimized convolution implementations, with no loss in accuracy.

1 INTRODUCTION

The rise in popularity of deep CNNs has spawned a research effort to find lower complexity networks to increase inference efficiency. This is *desirable* for inference in the cloud and becomes *crucial* on mobile and IoT devices with much more constrained hardware (Lane & Warden, 2018). Over the last few years, multiple approaches have been proposed to alleviate the compute-bound nature of convolutions (Sze et al., 2017). Arguably, the use of depthwise convolutions, as popularized by the family of MobileNet architectures (Howard et al., 2017; Sandler et al., 2018; Howard et al., 2019), has become the most widely embraced design choice to make lightweight networks. These layers are used in state of the art image classification networks (Stamoulis et al., 2019; Tan & Le, 2019). However, beyond image classification, normal convolutions are still chosen in favour of depthwise convolutions for applications like image super-resolution (Zhang et al., 2018; Lee et al., 2019), image segmentation (Takikawa et al., 2019) and, GANs (Brock et al., 2019). Therefore, alternative forms of speeding up standard convolutions are required to run these applications in mobile CPUs, which often come with constrained compute and energy budgets (Whatmough et al., 2019). Model quantization and the use of alternative convolution algorithms instead of direct convolution are two ways of accomplishing this task.

Lower-precision networks result in smaller model sizes, faster inference, lower energy consumption and smaller chip area (Sze et al., 2017; Li et al., 2019). Concretely, 8-bit quantizatized models achieve comparable performance to full-precision models (Jacob et al., 2018; Krishnamoorthi, 2018) while being ready for deployment on off-the-shelf hardware as 8-bit arithmetic is widely supported. In addition to resulting in a direct $4\times$ model size reduction, 8-bit integer-only arithmetic benefits from up to $116\times$ and $27.5\times$ chip area reduction compared to full precision additions and multiplies respectively, requiring $30\times$ and $18.5\times$ less energy (Horowitz, 2014; Whatmough et al., 2018). Because of these desirable benefits, 8-bit quantization has been widely adopted in both compute-constrained devices (Liberis & Lane, 2019; Chowdhery et al., 2019; Wang et al., 2019) and accelerators (Whatmough et al., 2019).

Orthogonal to lightweight architectural designs and quantization, fast convolution algorithms in replacement of direct

This project has received funding from the European Union's Horizon 2020 research and innovation programme under grant agreement No 732204 (Bonseyes). This work is supported by the Swiss State Secretariat for Education Research and Innovation (SERI) under contract number 16.0159. The opinions expressed and arguments employed herein do not necessarily reflect the official views of these funding bodies.

[1]Department of Computer Science, University of Oxford (UK) [2]Arm ML Research Lab. Correspondence to: Javier Fernandez-Marques <javier.fernandezmarques@cs.ox.ac.uk>.

Proceedings of the 3^{rd} MLSys Conference, Austin, TX, USA, 2020. Copyright 2020 by the author(s).

convolutions can provide further speedups. These come with their own trade-offs (Anderson & Gregg, 2018), but in this work we focus on the Winograd algorithm since it is the fastest known algorithm for convolutions of the dimensions often found in CNNs. The Winograd convolution performs the bulk of the computation as a Hadamard product between weights and input in the Winograd space requiring $\mathcal{O}(n)$ operations. Unlike normal convolutions, that generate a single output per convolution, a Winograd convolution computes several outputs simultaneously. This property makes Winograd convolutions minimal in the number of *general multiplications*[1] (Winograd, 1980). Normal convolutions operate on tile sizes matching the width and height of the filter, on the other hand, Winograd convolutions can operate on larger tiles. This is illustrated in Figure 1. In this way, while normal convolutions would require 8.1K multiplications to densely convolve a 32×32 input with a 3×3 filter, Winograd convolutions operating on a 4×4 tile require only 3.6K. The speedups Winograd convolutions offer increase with tile size. However, exploiting these speedups exposes a problem inherent in current Winograd convolution implementations: numerical error. This error, which grows exponentially with tile size, is the primary reason why Winograd is generally only deployed with 32-bit floating point and for comparatively small tile sizes, rarely larger than 6×6. In practice, an architecture with standard convolutional layers would first be trained before replacing standard convolution with Winograd convolution for deployment.

In this paper, we focus on alleviating the problem of numerical error that arises when using Winograd convolutions in quantized neural networks. Achieving this ultimately enables us to combine the speedups of Winograd with those that reduced precision arithmetic is known to offer, among other benefits in terms of energy and area. To this end, we present an end-to-end training pipeline that exposes the numerical inaccuracies introduced by Winograd to the learning of the model parameters. We also address the source of the numerical error and propose a relaxation on the form of the transformation matrices used in the Winograd convolution algorithm. We achieve this by adding these matrices to the set of *learnable* parameters in a layer, after initializing them via Cook-Toom (L. Toom, 1963). Finally, we describe wiNAS, a Winograd-aware Neural Architecture Search framework which leverages Winograd-aware layers and latency measurements on Arm Cortex-A73 and A53 cores, to jointly optimize for high accuracy and low latency. Our framework transforms a given macro-architecture by replacing each convolution with either *im2row* or Winograd convolutions of different tile sizes.

The contributions of this work are summarized below:

[1] *General multiplications* is a term commonly used in Winograd jargon referring to element-wise or Hadamard product stage.

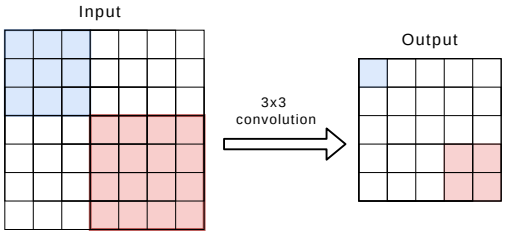

Figure 1. Standard convolutions operate on a tile (blue) defined by the filter size, generating a single output. Winograd convolutions operate on larger tiles without modifying the filter dimensions. A 3×3 Winograd convolution operating on a 4×4 tile (red) generates a 2×2 output. This is often expressed as $F(2 \times 2, 3 \times 3)$.

- We show that Winograd-aware networks enable Winograd convolutions in quantized networks, including 8-bit networks with little accuracy drop. To the best of our knowledge, this is the first time this has been empirically demonstrated.
- We demonstrate that learning the Winograd transforms, as opposed to keeping these fixed, results in better network accuracy – up to 10% improvement when using 6×6 and 8×8 tiles in 8-bits CNNs with 3×3 filters. This improvement is more pronounced with larger 5×5 filters.
- We present wiNAS as a tool that can find Winograd-aware networks jointly optimised for both high accuracy and low latency given a real hardware model.

2 Related Work

Convolutions have become the *de facto* spatial feature extractor in neural networks. As a result, a number of approaches have emerged to reduce the computational costs of using this operator.

Compact CNN Architectures. These include alternative formulations to the dense convolutional layer, such as *bottleneck* layers (He et al., 2016) that perform the 3×3 convolutions in a lower-dimensional space, or the *depth-wise* convolutional layers (Howard et al., 2017) which replace the standard convolutions with a channel-wise convolution followed by a 1×1 point-wise convolution. More recently, Chen et al. (2019) proposed a compact multi-resolution convolutional block that reduces spatial redundancy of low frequencies resulting in faster inference, memory savings and slightly higher accuracy. This reinforces the proposition that current networks rely more on texture than shape for image/object discrimination (Brendel & Bethge, 2019; Geirhos et al., 2019). In this work, instead of presenting a new architecture, we propose an optimization for an existing known-good architecture to speed up inference. Our

optimization can be applied to existing pre-trained models without the need for end-to-end training.

Quantization. The most extreme form of quantization is binary networks (Courbariaux et al., 2015; Lin et al., 2017; Xiang et al., 2017), which replace convolutions with bit-shifts resulting in $58\times$ inference speed-ups (Rastegari et al., 2016). Ternary and 2-bit models (Li et al., 2016; Wan et al., 2018; Gong et al., 2019) achieve higher accuracies while alleviating some the challenges of training binary networks (Alizadeh et al., 2019). However, it is 8-bit quantization (Jacob et al., 2018; Krishnamoorthi, 2018; Wang et al., 2019) that has achieved high popularity due to its balance between accuracy, model size reduction and inference speedup. Newer data formats, such as Posit (Carmichael et al., 2019) aim to close the accuracy gap between INT8 and FP32 networks, however hardware supporting it is unavailable. For training, BFLOAT16 (Kalamkar et al., 2019) has been validated as an alternative to FP32, enabling faster training. In this work, we adopt INT8 and INT16 uniform quantization during training and study how lowering precision impacts on the lossy nature of Winograd convolutions.

Fast Convolution Algorithms. Alternative formulations of the convolution operation such as: the use of FFTs, which replace convolution with its multiplication-only counterpart in the frequency domain resulting in faster inference (Mathieu et al., 2013; Abtahi et al., 2018) and training (Highlander & Rodriguez, 2015); the Strassen algorithm (Strassen, 1969), which when applied to convolutions (Cong & Xiao, 2014; Tschannen et al., 2018) significantly reduces the number of multiplications at the cost of more additions; or the Winograd algorithm (Winograd, 1980), which replaces convolutions with a set of matrix transformations and point-wise multiplications and, results in significantly faster inference stages (Lavin & Gray, 2016).

Winograd Convolution. The Winograd algorithm for fast convolution was first applied to CNNs by Lavin & Gray (2016), showing $2.2\times$ speedup compared to cuDNN (Chetlur et al., 2014) on a VGG (Simonyan & Zisserman, 2015) network, with no loss in accuracy on small 4×4 tiles, batch 1. However, exploiting Winograd convolutions on larger input tiles is challenging due to numerical instability. In response to this limitation, Barabasz et al. (2018) showed that the error introduced by the Winograd algorithm grows *at least exponentially* with tile size, which can be partially alleviated by choosing better polynomial points for constructing the transformation matrices via Cook-Toom (L. Toom, 1963). An alternative formulation using trimmed Vandermonde matrices was described by Vincent et al. (2017). More recently, several works studying the suitability of Winograd convolutions in memory and compute constrained setups have been proposed. These include: the use of integer arithmetic for complex Winograd convolutions (Meng & Brothers, 2019); a general formulation for the Winograd algorithm (Barabasz & Gregg, 2019) that shows promising results in FP16 and BFLOAT16 when using higher degree polynomials; an efficient region-wise multi-channel implementation of Winograd convolutions using General Matrix Multiplications (GEMMs) (Maji et al., 2019) that achieves $4\times$ speedups on Arm Cortex-A CPUs; and, a technique (Liu et al., 2018) that enables up to 90% sparsity in the Hadamard product stage of the Winograd algorithm, effectively reducing by $10\times$ the number of multiplications with no accuracy loss in FP32 models. Our work fills the gap of using Winograd convolutions in quantized neural networks, enabling even faster convolutions in current off-the-shelf hardware, such as mobile CPUs.

Neural Architecture Search. Automating the process of designing neural network architectures has drawn considerable attention. Early attempts relied on reinforcement learning (Zoph & Le, 2017; Brock et al., 2018; Real et al., 2019; Tan et al., 2019) or Bayesian optimization (Hernández-Lobato et al., 2016; Fedorov et al., 2019) and required thousands of GPU hours to converge due to their computationally expensive and exhaustive search stages. Other works opted instead for a gradient-based search by framing the problem as a single over-parameterized network where all *candidate operations* at a particular node (e.g. a layer) are taken into consideration. The main aspect differentiating gradient-based NAS approaches is the way the output of a layer combines the contribution of each candidate operation. While Bender et al. (2018) defines it as the sum and DARTS (Liu et al., 2019) as a weighted sum, ProxylessNAS (Cai et al., 2019) relies on path-level binarization, making it possible to perform the search on the entire architecture directly using a single GPU. In addition to architecture discovery, NAS has also been successfully used for automated network pruning (He et al., 2018) and quantization (Wang et al., 2019). Our work leverages NAS to find the optimal convolution algorithm (i.e. *im2row* or different Winograd implementations) for each layer in the model while preserving the overall network macro-architecture and model size.

3 Winograd-Aware Networks

This section introduces Winograd convolutions and their trade-offs in terms of compute, memory and accuracy. Then, we present the Winograd-aware layers used in our networks.

3.1 Winograd implementation trade-offs

The Winograd algorithm for convolutions using linear polynomials guarantees to use the minimum number of element-wise multiplications to compute $m \times m$ outputs using an $r \times r$ filter. Lavin & Gray (2016) refer to this minimal

algorithm as $F(m \times m, r \times r)$ and present its matrix form:

$$Y = A^\top \Big[[GgG^\top] \odot [B^\top dB] \Big] A \tag{1}$$

where G, B and A are transformation matrices applied to the filter g, input and output respectively and $\odot$ is the Hadamard or element-wise multiplication.

These transformation matrices are commonly constructed[2] as described in the Cook-Toom algorithm which requires choosing a set of so-called *polynomial points* from $\mathbb{R}^2$. This choice is not trivial, but for small Winograd kernels e.g. $F(2 \times 2, 3 \times 3)$ or $F(4 \times 4, 3 \times 3)$, there is a common consensus. While a standard convolution using a $r \times r$ filter g would operate on a $r \times r$ input tile, a Winograd convolution expressed as Eq. 1 expects an input patch d with dimensions $(m+r-1) \times (m+r-1)$. The key difference is that while the standard convolution would generate a 1×1 output, the Winograd convolution would compute a $m \times m$ output. In this way, a standard 3×3 convolution requires 9 mult. per output (*mpo*), $F(2 \times 2, 3 \times 3)$ and $F(4 \times 4, 3 \times 3)$ require 4 *mpo* and 2.25 *mpo* respectively. Theoretically, these savings grow as we increase the tile or filter sizes. For the remaining of this work and, unless stated otherwise, we will be considering 3×3 filters and therefore refer to $F(2 \times 2, 3 \times 3)$ as $F2$, $F(4 \times 4, 3 \times 3)$ as $F4$, and so on.

The challenges associated with the use of Winograd convolutions span three dimensions:

Compute. Winograd convolutions require the transformation of both tile d and filter g to the Winograd domain. The cost of these transformations grows with m, and can represent a significant portion of the total computation of up to 75% (Sec. 6.2). This suggests that Winograd offers little to no speedup in layers with few filters. The cost of $GgG^\top$ is often ignored as it is amortized across inferences.

Memory. In Eq.1, $GgG^\top$ transforms the filter g to the Winograd domain, matching the dimensions of the input tile d. This results in an increase of run-time memory associated with the weights: $1.78\times$ and $4\times$ for $F2$ and $F4$ respectively. This is especially undesirable on memory-constrained devices such as microcontrollers.

Numerical Error. Small $F2$ and $F4$ perform well in single and double precision (FP32/64) networks and are available in production-ready libraries such as NVIDIA cuDNN (Chetlur et al., 2014) and Arm Compute Library (Arm Software). Because these introduce only marginal numerical error, a network can first be trained using conventional convolutions before replacing appropriate layers with Winograd, without impacting accuracy.

[2] See Section 5.2 in Blahut (2010) for a step-by-step example.

	ResNet-18 Accuracy		
Convolution method	32-bit	16-bit	8-bit
Direct	93.16	93.60	93.22
Winograd $F2$	93.16	93.48	93.21
Winograd $F4$	93.14	19.25	17.36
Winograd $F6$	93.11	11.41	10.95

Table 1. Replacing the convolutional layers in pre-trained ResNet-18 models on CIFAR-10 with $F2$, $F4$ and $F6$. This works well in full precision, but accuracy drops drastically with quantization for configurations beyond $F2$. Note that prior to evaluating the quantized configurations we performed a *warmup* of all the moving averages involved in Eq.1 using the training set but without modifying the weights. Without this relaxation (which requires a Winograd-aware pipeline as in Fig. 2), $F2$ would be unusable.

However, attempting this with larger Winograd tiles, or in combination with quantization, results significant accuracy loss. The root of the problem[3] is the increasing numerical range in G, B and A as d increases. As a consequence, the multiple matrix multiplications in Eq.1 contribute considerable error, ultimately reducing accuracy. This problem is exacerbated in networks using quantized weights and activations, where the range and precision of values is reduced. We show these limitations in Table 1. The numerical error is, to a large extent, the main limiting factor for adopting large-tiled Winograd and for adopting Winograd convolutions in general for reduced precision networks.

In this work we focus on minimizing the numerical errors that arise when using the Winograd algorithm in quantized networks. Our approach does not aggravate the compute and memory challenges previously mentioned. Instead, it indirectly alleviates these by making use of quantization.

3.2 A Winograd-aware training pipeline

Neural networks have proven to be resilient to all kinds of approximations, e.g. pruning and quantization. When applying these techniques, consistently better models are generated if these approximations are present during training. In other words, when the training is *aware* of quantization, or when training is *aware* of pruning.

Following this intuition, we propose an end-to-end Winograd-aware pipeline as shown in Figure 2. In the forward pass we apply Eq.1 to each patch of the activations from the previous layer. We can apply standard back-propagation, since Eq.1 is only a collection of matrix-matrix multiplications. This implementation allows us to:

- **Learn better filters.** Building an explicit implementation of each of the stages involved in the Winograd

[3] We refer the interested reader to Barabasz et al. (2018) for an analysis on the nature of the errors in Winograd convolutions.

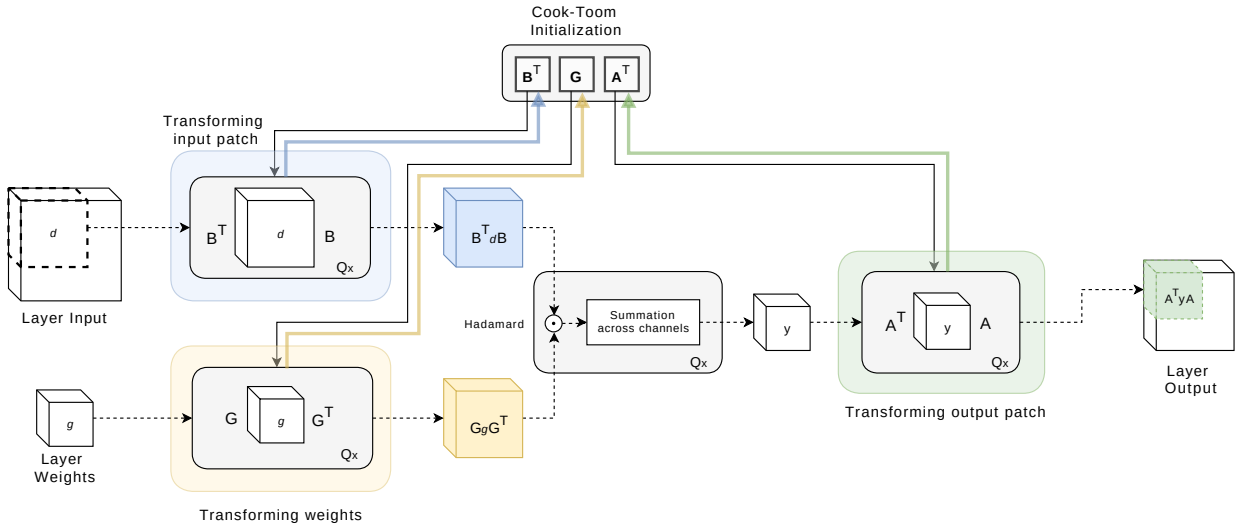

Figure 2. Forward pass of Winograd-aware layers. Transformation matrices G, $B^\top$ and $A^\top$ are constructed via Cook-Toom. If these are included in the set of model parameters, they would be updated with every batch via back-progation (this is represented with the coloured arrows going back to matrices G, $B^\top$ and $A^\top$, carrying the gradients to update the values of each transform). In its default configuration, each intermediate output throughout the pipeline quantized to the same level as the input and weights, this is represented by Qx.

transform exposes the numerical errors introduced in Eq.1 to the learning of the filters. This prevents the accuracy drops shown in Table 1.

- **Learn the transforms.** Traditionally, matrices G, $B^\top$ and $A^\top$ are fixed. Instead, we can treat them as another set of *learnable* parameters in the layer. This relaxation leads to much improved performance in quantized networks while still maintaining the overall structure of the Winograd convolution algorithm and its speedups.
- **Quantization diversity.** Unlike standard convolution, which does not require intermediate computation, Winograd convolution requires at least four of them for $GgG^\top$, $B^\top dB$, the Hadamard product and the output transformation. Each of these can be quantized to a different number of bits depending on the bit-width of the input, that of the weights, and the overall complexity of the problem the network is designed to solve.

4 Searching for Winograd-Aware Networks

Simultaneously maximizing accuracy and minimizing latency with Winograd convolution isn't trivial. The reason for this is that large tiles result in low latency, but come at the cost of higher numerical error. This presents a good opportunity to jointly optimize network accuracy and latency.

To this end, we implement a NAS-based approach that automatically transforms an existing architecture into a Winograd-aware version. We perform NAS at the micro-architecture level by selecting from different convolution algorithms for each layer, but without modifying the network's macro-architecture (e.g. number or order of layers, hyper-parameters, etc). Keeping the macro-architecture fixed allows us to fairly compare the standard model to its Winograd-aware counterpart in terms of latency and accuracy. We call our framework wiNAS.

4.1 Winograd-aware NAS pipeline

Introducing latency measurements into the optimization objective requires knowing the shape of the input tensor, i.e. the activations from the previous layer, at each layer of the network. We design wiNAS as a variation of ProxylessNAS (Cai et al., 2019), leveraging path sampling while performing the search. This technique, enables the allocation of the entire network on a single GPU by evaluating no more than two candidate operations at each layer per batch.

Similarly to ProxylessNAS, wiNAS formulates the search as a two-stage process, alternating the update of model parameters (the weights), where the loss is defined as

$$L_{weights} = Loss_{CE} + \lambda_0 \|w\|_2^2 \tag{2}$$

and the update of architecture parameters (the weight assigned to each operation on a given layer), where the loss introduces the latency metrics is defined as

$$L_{arch} = Loss_{CE} + \lambda_1 \|a\|_2^2 + \lambda_2 E\{latency\} \tag{3}$$

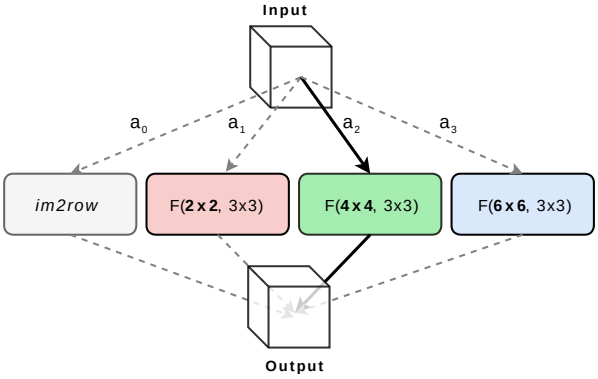

Figure 3. With wiNAS, each 3×3 convolution in a given architecture is implemented with either *im2row* or with Winograd convolutions varying tile size. While the former is lossless and faster than direct convolution, Winograd offers lower latencies but introduce numerical instability that could ultimately impact in accuracy.

where a are the architecture parameters and λ_2 controls the impact of latency in the loss. The expected latency, $E\{latency\}$, for a given layer is the weighted combination of the latency estimate of each candidate operation with their respective probability of being sampled. Intuitively, searching for Winograd convolutions with high λ_2 would result in faster models, potentially at the detriment of accuracy.

Unlike ProxylessNAS, wiNAS focuses on simply selecting the optimal convolution algorithm for each of the 3×3 convolutional layers. Therefore, the set of candidate operations for a given *conv2d* layer contains *im2row* and Winograd-aware layers in their $F2$, $F4$ and $F6$ configurations. This search space is illustrated in Figure 3. Each candidate operation comes with its respective latency, which is a function of the output dimensions and quantization level.

5 Experimental Setup

We conduct various experiments grouped in three categories. In this section we describe each experiment we conducted. We used PyTorch (Paszke et al., 2017) for training and Arm Compute Library for deployment.

5.1 Vanilla Winograd-aware networks

We begin our study of Winograd-aware networks by performing an extensive evaluation on the ResNet-18 (He et al., 2016) architecture using the CIFAR-10 (Krizhevsky et al., 2009) dataset. In this experiment we train the network end-to-end using standard convolutions, and $F2$, $F4$ and $F6$ Winograd convolutions. For each experiment with Winograd, all layers in the network use the same tile size, except the last two residual blocks which are kept fixed to $F2$.The input convolutional layer uses normal convolutions. We run the experiments for FP32, INT16, INT10 and INT8 quantized networks, where both weights and activations are uniformly quantized (including all the intermediate outputs shown in Figure 2). We follow the per-layer symmetric quantization as described in Krishnamoorthi (2018).We repeated each experiment while enabling the Winograd transforms G, $B^\top$ and $A^\top$ to be learnt, which we denote using the additional suffix *-flex*.

Winograd-aware layers do not require an over-parameterized model to perform well. We also varied the model size by using a width-multiplier, as used by the MobileNets family, ranging from 0.125 to 1.0, meaning that when the multiplier is 1.0 the network is the full ResNet-18. This leads to models ranging between 215K and 11M parameters. Winograd-aware layers with learnable transformations marginally increase ($< 0.1\%$) the model size, since the transforms themselves need to be saved for model deployment. We repeated the experiment for CIFAR-100 (Krizhevsky et al., 2009), but without varying the depth-multiplier. CIFAR-100 is considerably more challenging that CIFAR-10, as it is comprised of 100 classes with only 600 images per class.

Additionally, we use an INT8 LeNet (Lecun et al., 1998), trained on the MNIST dataset, to evaluate the suitability of Winograd-aware layers with learnable transforms for 5×5 filters. This is a more challenging case than 3×3 filters, because a larger tile tile is required (defined by $F(m \times m, r \times r)$), with larger transformation matrices which require the choice of more good polynomial points.

For experiments on ResNet-18, we replace 2×2-stride convolution layers with a 2×2 max-pooling layer followed by a dense 3×3 convolution layer. Altering the network in this way is necessary since there is no known equivalent for strided Winograd convolutions, which remains an open research question. This is a common strategy when evaluating Winograd (Liu et al., 2018; Choi et al., 2018). We also modified the number of output channels of the input layer from 64 to 32. We did this to reduce the memory peak during training. We use the Adam (Kingma & Ba, 2015) optimizer and train for 120 epochs. Both CIFAR-10/100 use the same ResNet-18 architecture, differing only in the number of outputs of the fully connected layer. Results for other architectures are shown in A.1.

5.2 wiNAS: Winograd-aware NAS

To evaluate wiNAS, we define two different sets of *candidate operations*. These spaces are: wiNAS_{WA} and $\text{wiNAS}_{\text{WA-Q}}$, both allowing each 3×3 convolutional layer to be implemented with either *im2row* or each of the Winograd configurations, $F2$, $F4$ or $F6$. The former uses a fixed bit-width for all elements in the architecture, while the latter introduces in the search space candidates of each operation quantized to FP32, INT16 and INT8.

The hyperparameters used for wiNAS are as follows: for the learning of model parameters we use mini-batch SGD with Nesterov momentum (Sutskever et al., 2013). In the stage where we update the architecture parameters we use instead Adam with the first momentum scaling, β_1, set to zero, so the optimizer only updates paths that have been sampled. For both stages we use Cosine Annealing (Loshchilov & Hutter, 2017) scheduling and a batch size of 64. We perform the search for 100 epochs in each search space at different λ_2 values ranging from 0.1 to 1e-3. Once the search is completed, we trained the architecture end-to-end with the same hyperparameters as the rest of winograd-aware networks.

5.3 Winograd convolutions on mobile CPUs

For our study, we chose Arm A73 and A53 cores on a Huawei HiKey 960 development board with the big.LITTLE CPU architecture. These cores are good candidates for validating the speedups that are achievable with Winograd convolutions in today's off-the-shelf mobile hardware.

CPU	Clock	L1	L2
A73	2.4 GHz	64 KB	2048 KB
A53	1.8 GHz	32 KB	512 KB

Table 2. Key hardware specifications for the high-performance Cortex-A73 and the high-efficiency Cortex-A53 cores found on a HiKey 960 development board.

While both A73 and A53 are implemented as 16nm quad-core CPUs, the former is a high-performance processor and the latter implements a high-efficiency processor. In Table 2 we summarise the main differences between these CPUs. The memory bandwidth would be the primary factor that ultimately sets the upper limit to the speedup achievable by Winograd since it requires operating in larger tiles than direct convolution algorithms such as *im2row* or *im2col*.

In our study, we measured the time taken for 3×3 convolutions using *im2row*, *im2col* and each of the Winograd configurations ($F2$, $F4$, $F6$) when varying output width/height (from 112×112 down to 2×2) and $inCh \rightarrow outCh$ (from $3 \rightarrow 32$ to $512 \rightarrow 512$). We performed the benchmark in controlled conditions and in single thread mode. Each combination was run five times with five seconds delay in between to prevent thermal throttling. We implemented Winograd convolutions using GEMMs (Maji et al. (2019)), and performed the same experiment separately on A73 and A53 for both FP32 and INT8. INT16 measurements are not currently supported in Arm Compute Library.

6 Experimental Results

The results of this work are arranged as three subsections. First, we show that winograd-aware networks can achieve high accuracy. Second, we present the results from our dense benchmark for winograd convolutions on mobile CPUs. Third, we show that wiNAS can jointly optimize a given macro-architecture for accuracy and latency.

6.1 Vanilla Winograd-aware networks

Figure 4 (left) shows Winograd-aware networks in FP32 perform as well as direct convolutions, with both fixed and learned (*-flex*) transformation matrices. With quantization (all other plots), winograd-aware layers are essential to enable the use of fast Winograd convolutions. This is not possible if switching to Winograd convolutions after training, as is commonly done in practice (see Table 1).

Furthermore, we show that learning the Winograd transforms (*-flex*) results in 10% and 5% better accuracies for $F4$ and $F6$ in INT8 scenarios. We argue that enabling this relaxation helps in easing the numerical instability inherent to Winograd convolutions, which is further exacerbated by quantization. The accuracy of Winograd-aware models scales linearly with network width, suggesting that these can be exploited in conjunction with architecture compression techniques such as channel pruning.

Results from LeNet (5×5 filters), provides further evidence that larger tiles result in higher numerical error. In Figure 5, we show that even in relatively small datasets like MNIST, keeping the transformations G, $B^\top$ and $A^\top$ fixed, leads to poor results as the output tile size is increased. This difference is almost 47% in the case of $F(6 \times 6, 5 \times 5)$ layers, which uses 10×10 tiles.

Winograd-aware layers do not structurally modify the network architecture, since Winograd is just an algorithm to perform convolution. We demonstrate it is possible to transform a pre-trained model with standard convolution into its Winograd-aware counterpart within a few epochs. Concretely, in Figure 6 we show that an INT8 ResNet-18 $F4$ can be adapted from a model of the same network that was trained end-to-end with standard convolutions in 20 epochs of retraining. This represents a $2.8\times$ training time reduction for Winograd-aware models. This is only possible when allowing the transformation matrices to evolve during training. Adapting FP32 models can be done in a single epoch.

We believe both $F4$ and $F6$ performance could be raised with alternative quantization implementations, closing the accuracy gap with $F2$ and direct convolutions.

6.2 Impact of Winograd on Latency

The speedups associated to with use of Winograd convolutions often only account for the point-wise stage while assuming negligible costs for the input, weights and output transformations. Furthermore, these also assume that the larger the input patch, *d*, the larger the speedup compared to

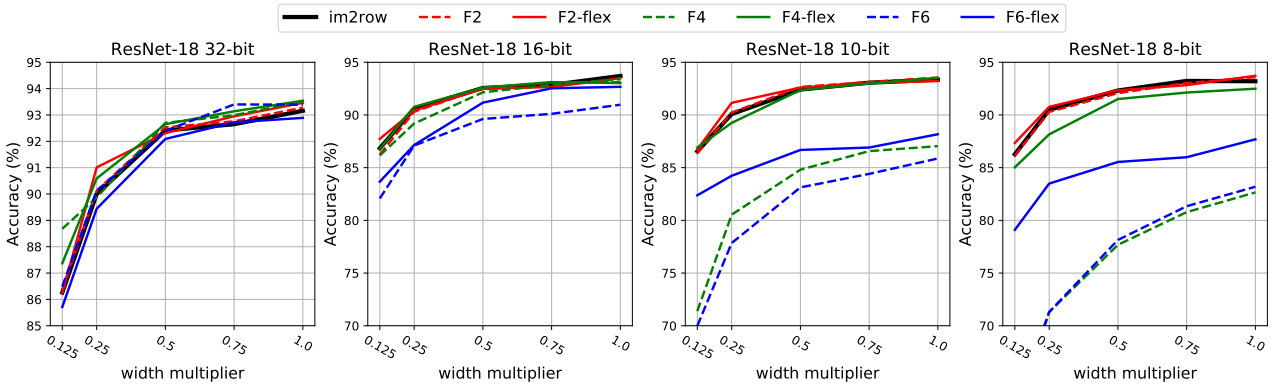

Figure 4. Performance of a winograd-aware ResNet-18 at different bit-widths and trained with different Winograd configurations. We show how winograd-aware layers scale with network's width. We can observe that in quantized networks, models that learn the Winograd transforms (*-flex* configurations), strictly outperforms those models that keep them fixed with the values obtained via Cook-Toom.

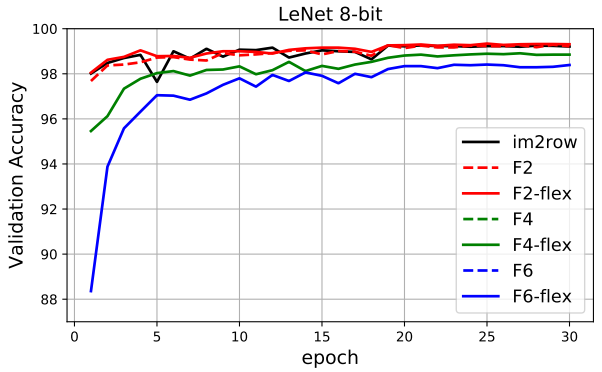

Figure 5. Performance of INT8 LeNet on MNIST using standard convolutions (*im2row*) or winograd-aware layers. Letting the transformations to evolve during training (*-flex*) always results in better models. F4 and F6 configurations (not shown) reach an accuracy of 73% and 51%, respectively. All configurations reach $99.25\% \pm 0.1\%$ in full precision.

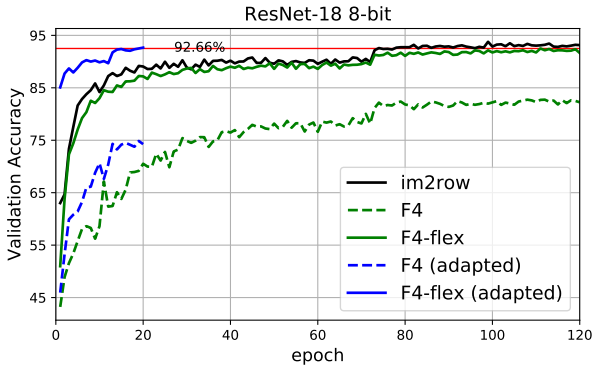

Figure 6. Transforming a standard model (trained with default convolutions) to its Winograd-aware counterpart can be done in very few epochs of retraining. We found this technique works best if the Winograd transformations are learnt during retraining (*-flex*), otherwise adaptation becomes much more challenging.

normal convolutions. However, although these assumptions are true for large tensors, they are not necessarily true when working with tensors of the sizes often found in CNNs for image classification or object detection.

Figure 7 shows a small portion of the obtained latency measurements for our benchmark in FP32. An almost identical pattern appears when using 8-bit arithmetic. In Figure 8 we show the speedups that Winograd convolutions offer at different layers of a ResNet-18 network. Our observations can be summarized in three points:

Input layers do not benefit from Winograd. This is primarily because the matrices in the element-wise GEMM stage are not large enough to compensate for the costs of transforming the input and output patches (see Figure 5.3 and 8). They represent a significant portion (up to 65% and 75% respectively on the A73 and A53) of the total costs for convolving a RGB 32×32 input expanded to 32 channels. Similar ratios can be observed for other input sizes. In spite of this, this first layer accounts for a marginal portion of the total latency of the model, often below 1ms.

Optimal $\mathbf{m}$ is a function of input width and height. For an input with sufficient number of channels, e.g. 32 channels and up, we observe a consistent pattern alternating between $F4$ and $F6$ as the channel dimension of the output increase. This alternation comes as a result of the impossibility of subdividing the input into an integer number of $(m+r-1) \times (m+r-1)$ patches, and therefore having to waste calculations when operating around the matrix edges. This pattern is invariant to different $inCh \rightarrow outCh$ configurations and fades away as input dimensions exceed 40×40, where F6 consistently becomes the fastest.

Winograd transforms are costly. Excluding inputs with very few channels, the cost of performing the transformations to/from the Winograd domain can exceed 25% of the overall costs. These costs become negligible as the input width and height decrease, but the rate at which this happens also depends on the hardware. Our Winograd-aware

	inCh --> outCh																			
	3 --> 32				32 --> 64				128 --> 192				192 --> 256				256-->512			
outW	im2row	F2	F4	F6	im2row	F2	F4	F6	im2row	F2	F4	F6	im2row	F2	F4	F6	im2row	F2	F4	F6
2	0.007	0.008	0.016	0.029	0.070	0.043	0.082	0.167	0.659	0.407	1.219	2.196	1.463	1.082	2.378	4.407	3.912	2.932	6.619	11.853
4	0.011	0.029	0.016	0.030	0.154	0.078	0.081	0.167	1.642	0.802	1.170	2.195	2.884	1.731	2.502	4.486	7.450	4.962	6.588	11.947
6	0.021	0.053	0.065	0.029	0.328	0.199	0.174	0.165	4.137	2.229	2.040	2.148	6.780	4.559	4.135	4.327	17.450	13.858	11.452	11.919
8	0.031	0.059	0.064	0.133	0.519	0.280	0.175	0.408	5.306	2.993	2.004	3.899	10.932	6.145	4.167	7.907	28.238	14.930	11.499	21.241
10	0.058	0.101	0.119	0.144	0.910	0.475	0.482	0.412	9.466	5.054	5.321	3.973	17.808	10.198	10.318	7.904	44.656	27.597	32.685	21.437
12	0.066	0.133	0.129	0.132	1.208	0.621	0.475	0.424	11.625	6.601	5.382	3.971	24.196	12.995	10.272	7.955	61.236	35.702	32.164	21.478
14	0.087	0.186	0.154	0.267	1.610	0.868	0.695	1.043	16.177	9.277	7.498	9.846	33.702	18.154	14.220	19.082	85.809	48.590	34.306	60.003
16	0.111	0.235	0.153	0.283	2.592	1.191	0.723	1.051	20.845	12.158	7.551	10.002	42.362	23.147	14.310	19.263	109.943	57.083	34.190	60.504
18	0.169	0.281	0.263	0.281	3.315	1.379	1.133	1.031	26.785	15.125	12.159	9.961	55.085	29.292	23.178	19.476	142.460	75.505	63.799	60.987
20	0.184	0.325	0.249	0.400	3.416	1.695	1.131	1.728	32.851	18.450	12.115	15.108	67.300	35.276	23.274	27.723	173.488	90.041	65.349	67.923
22	0.210	0.398	0.331	0.410	4.164	2.070	1.506	1.690	40.245	22.207	16.010	15.114	82.028	43.166	30.697	27.781	213.326	110.160	82.434	67.228
24	0.247	0.452	0.324	0.409	4.783	2.453	1.498	1.729	47.961	26.600	16.126	15.035	97.706	51.064	30.954	27.923	251.771	125.604	83.167	67.047

Figure 7. Latencies (in milliseconds) of convolving increasingly larger input tensors in the width/height dimensions (y axis) and in depth (x axis). We compare the time needed for *im2row* and each of the Winograd configurations with 32-bit arithmetic on a Cortex-A73. We show that (1) *im2row* is the consistently the optimal algorithm for the input layer to a network, (2) the choice between $F2$, $F4$ and $F6$ should be done based on the output's width/height and, (3) this choice should not generally be altered based on $inCh \rightarrow outCh$.

pipeline formulation doesn't impose any constrains on how the transforms are learnt. This results in dense transforms (as opposed to the default transforms witch contain zeros) and therefore applying them require additional compute. Table 3 includes this latency overhead in models making use of learned transforms. In Appendix A.2 we provide more details on how dense transforms impact overall latency.

On A53, the speedups from FP32 Winograd convolutions are smaller than On A73. We argue this comes as a results of the differences in the memory subsystem, limiting the lower-end CPU to efficiently operate with larger tensors. These speedups grow significantly when leveraging INT8 arithmetic, made possible by winograd-aware training. Concretely, INT8 Winograd increases the speedup on the A53 by a factor of almost $1.5\times$ compared to Winograd in FP32, as shown in WA_{F4} configurations in Table 3 – at the cost of 1.1% accuracy in CIFAR-10. In the case of the more challenging CIFAR-100 dataset, the drop in accuracy is more severe. However, our WA_{F2} layers offer attractive speedups for INT8 with no drop in accuracy. We rely on wiNAS to minimize this degradation with small impact on latency.

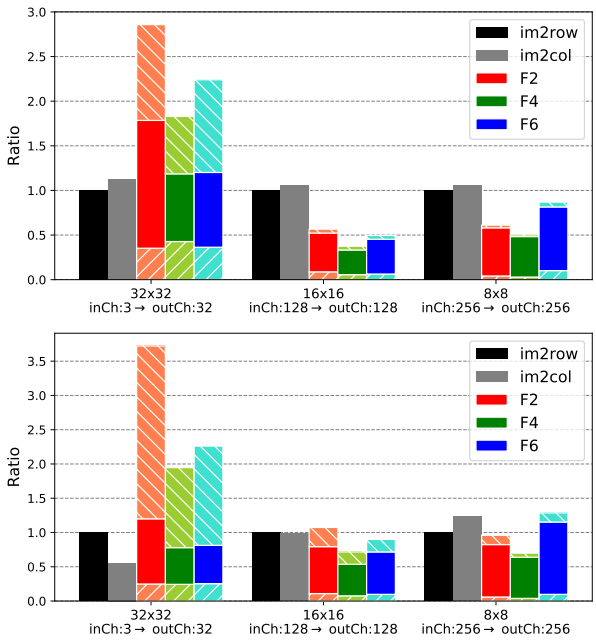

Figure 8. Latencies (normalized *w.r.t im2row*) to execute different layers of the ResNet-18 network measured in A73 (above) and A53 (below). For Winograds, solid colour bar regions represent the element-wise GEMM stage, below and above it are respectively the input and output transformation costs. Measured for FP32 and with default Winograd transforms.

6.3 wiNAS Networks

Choosing the convolution algorithm that minimizes overall network latency can be easily done by looking at the benchmark results. However, since the accuracy of Winograd convolutions degrade rapidly in reduced precision networks, selecting the fastest algorithm for each layer without sacrificing accuracy, is not straight forward.

When using wiNAS_{WA}, values of λ_2 larger than 0.05 consistently result in models with the same layer configuration as those in WA_{F4} (described in section 5.1). When lowering the impact of latency in Eq. 3 loss function, we observed several $F4$ Winograd-aware layers were replaced with either *im2row* or $F2$, at the cost of less than 9 ms latency increase in the worst case, an INT8 model on the A53 for CIFAR-100. These models resulted in similar accuracies in FP32 and reached 0.32% and 1.1% higher accuracies in INT8 for CIFAR-10 and CIFAR-100 respectively. Despite CIFAR-100 models sacrificing more latency in order to recover accuracy, they remained faster than WA_{F2} at INT8.

When introducing quantization in the search performed by $\text{wiNAS}_{\text{WA-Q}}$, the accuracy gap is almost closed for CIFAR-10 and further reduced for CIFAR-100. This comes primar-

Conv. Type	Bits act. / param.	Accuracy (%) CIFAR-10	CIFAR-100	Cortex-A53 Latency (ms)	Speedup	Cotex-A73 Latency (ms)	Speedup
im2row		93.16	74.62	118	-	85	-
im2col		93.16	74.62	156	0.76×	102	0.83×
W_{F2}		93.16	74.60	126	0.94×	56	1.52×
W_{F4}	32 / 32	93.14	74.53	97	1.22×	46	1.85×
WA_{F2}*		93.46	74.69	126	0.94×	56	1.52×
WA_{F4}		93.54	74.98	122†	0.92×	54†	1.58×
wiNAS_{WA}		93.35	74.71	123†	0.96×	56†	1.52×
im2row		93.20	74.11	117	1.01×	54	1.57×
im2col		93.20	74.11	124	0.95×	59	1.45×
WA_{F2}*	8 / 8	93.72	73.71	91	1.30×	38	2.24×
WA_{F4}		92.46	72.38	82†	**1.44×**	35†	**2.43×**
wiNAS_{WA}		92.71	73.42	88† / 91†	1.34× / 1.30×	35† / 36†	2.43× / 2.36×
$\text{wiNAS}_{WA\text{-}Q}$	auto	92.89	73.88	74† / 97†	**1.60×** / 1.22×	32† / 43†	**2.66×** / 1.98×

Table 3. Performance in terms of accuracy and latency (ms) of ResNet-18 when convolutions are implemented with different algorithms and for different quantization levels. We show that Winograd-aware ($WA_{F2/4}$) layers combine the speedups of Winograd convolutions with those of INT8 arithmetic, with little to no accuracy loss in some cases. This is not possible with existing Winograd ($W_{F2/4}$) formulations. Latency is measured on Arm Cortex-A73 and A53 cores. For the last two rows, wiNAS found different optimizations for each dataset. We show latencies for CIFAR-10 on the left and CIFAR-100 on the right. Speedups are shown against *im2row* in FP32. (*) With default Winograd transforms. (†) Includes worst case latency increase due to be using learned transform, which are often dense.

ily as a result of relying on higher bit-widths for the first layers in the network. In both cases, we maintain attractive speedups compared to *im2row* and Winograd convolutions in FP32, specially on the A73. All the ResNet-18 architectures optimized with wiNAS are described in A.3.

7 Discussion

In this section we present some of the challenges of training Winograd-aware networks and propose lines of future work.

A direct implementation of Eq. 1 requires saving the intermediate outputs of each matrix-matrix multiplication, since these are needed for back-propagation. This results in high memory usage. In this work we had to rely on *gradient checkpointing* (Chen et al., 2016) to lower the memory peak during training, at the cost of additional computation. We believe a native CUDA implementation of the Winograd-aware layers with better memory reuse would ease this problem.

Learning larger models (with width multipliers 0.75 and 1.0) proved challenging for $F4$ and $F6$ when introducing quantization. Using other types of quantization would likely help. In particular per-channel affine quantization, as in Jacob et al. (2018). Also, enabling different bit-widths throughout Eq. 1 could help mitigate the accuracy drop.

It is known that bad polynomial points for constructing G, $B^\top$ and $A^\top$ introduce significant deviations in the result of computing Winograd convolutions compared to that of normal convolutions. We observed that good starting points are also important even when learning the Winograd transformations. Polynomial points specifically tailored for quantized Winograd could alleviate some of the degradation that we observed with increased tile size.

In this work we focused on mobile CPUs, but we expect these benefits to be also applicable to GPUs. However, to further maximize the speedups that Winograd-aware layers for quantized CNNs offer, a custom hardware implementation in the form of an accelerator would be preferable.

8 Conclusion

Running CNN-based applications that require standard convolutional layers is challenging in compute-constrained devices such as mobile CPUs. This paper presents Winograd-aware layers as the building block to combine the benefits of quantized networks and fast Winograd convolutions. We studied Winograd-aware layers with different tile sizes, three quantization levels and on three popular datasets. We found that allowing the transformation matrices to evolve during training resulted in significantly better models. With wiNAS we leveraged Winograd-aware layers and latency metrics from off-the-shelf mobile CPUs and found architectures that helped minize the numerical instability of Winograd. A Winograd-aware ResNet-18 quantized to INT8 offers up to 1.32× faster inference for only a marginal accuracy drop compared to existing Winograd implementations, which are limited to FP32. This network is also 1.54× faster than an optimized *im2row* implementation using INT8 arithmetic.

References

Abtahi, T., Shea, C., Kulkarni, A., and Mohsenin, T. Accelerating

convolutional neural network with fft on embedded hardware. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 26(9):1737–1749, Sep. 2018. ISSN 1063-8210. doi: 10.1109/TVLSI.2018.2825145.

Alizadeh, M., Fernandez-Marques, J., Lane, N. D., and Gal, Y. A systematic study of binary neural networks' optimisation. In *International Conference on Learning Representations*, 2019.

Anderson, A. and Gregg, D. Optimal dnn primitive selection with partitioned boolean quadratic programming. *Proceedings of the 2018 International Symposium on Code Generation and Optimization - CGO 2018*, 2018. doi: 10.1145/3168805.

Arm Software. Arm compute library. https://developer.arm.com/ip-products/processors/machine-learning/compute-library. Accessed: 2019-08-01.

Barabasz, B. and Gregg, D. Winograd convolution for dnns: Beyond linear polynomials, 2019.

Barabasz, B., Anderson, A., and Gregg, D. Improving accuracy of winograd convolution for dnns. *CoRR*, abs/1803.10986, 2018.

Bender, G., Kindermans, P.-J., Zoph, B., Vasudevan, V., and Le, Q. Understanding and simplifying one-shot architecture search. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 550–559, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Blahut, R. E. *Fast Algorithms for Signal Processing*. Cambridge University Press, 2010. doi: 10.1017/CBO9780511760921.

Brendel, W. and Bethge, M. Approximating CNNs with bag-of-local-features models works surprisingly well on imagenet. In *International Conference on Learning Representations*, 2019.

Brock, A., Lim, T., Ritchie, J., and Weston, N. SMASH: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*, 2018.

Brock, A., Donahue, J., and Simonyan, K. Large scale GAN training for high fidelity natural image synthesis. In *International Conference on Learning Representations*, 2019.

Cai, H., Zhu, L., and Han, S. ProxylessNAS: Direct neural architecture search on target task and hardware. In *International Conference on Learning Representations*, 2019.

Carmichael, Z., Langroudi, H. F., Khazanov, C., Lillie, J., Gustafson, J. L., and Kudithipudi, D. Deep positron: A deep neural network using the posit number system. *2019 Design, Automation and Test in Europe Conference and Exhibition (DATE)*, Mar 2019. doi: 10.23919/date.2019.8715262.

Chen, T., Xu, B., Zhang, C., and Guestrin, C. Training deep nets with sublinear memory cost, 2016.

Chen, Y., Fan, H., Xu, B., Yan, Z., Kalantidis, Y., Rohrbach, M., Yan, S., and Feng, J. Drop an octave: Reducing spatial redundancy in convolutional neural networks with octave convolution, 2019.

Chetlur, S., Woolley, C., Vandermersch, P., Cohen, J., Tran, J., Catanzaro, B., and Shelhamer, E. cudnn: Efficient primitives for deep learning. *CoRR*, abs/1410.0759, 2014.

Choi, Y., El-Khamy, M., and Lee, J. Compression of deep convolutional neural networks under joint sparsity constraints, 2018.

Chowdhery, A., Warden, P., Shlens, J., Howard, A., and Rhodes, R. Visual wake words dataset, 2019.

Cong, J. and Xiao, B. Minimizing computation in convolutional neural networks. In Wermter, S., Weber, C., Duch, W., Honkela, T., Koprinkova-Hristova, P., Magg, S., Palm, G., and Villa, A. E. P. (eds.), *Artificial Neural Networks and Machine Learning – ICANN 2014*, pp. 281–290, Cham, 2014. Springer International Publishing. ISBN 978-3-319-11179-7.

Courbariaux, M., Bengio, Y., and David, J.-P. 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*, pp. 3123–3131. Curran Associates, Inc., 2015.

Fedorov, I., Adams, R. P., Mattina, M., and Whatmough, P. N. SpArSe: Sparse Architecture Search for CNNs on Resource-Constrained Microcontrollers. In *Proceedings of the Neural Information Processing Systems (NeurIPS) Conference 2019*, 2019.

Geirhos, R., Rubisch, P., Michaelis, C., Bethge, M., Wichmann, F. A., and Brendel, W. Imagenet-trained CNNs are biased towards texture; increasing shape bias improves accuracy and robustness. In *International Conference on Learning Representations*, 2019.

Gong, R., Liu, X., Jiang, S., Li, T., Hu, P., Lin, J., Yu, F., and Yan, J. Differentiable soft quantization: Bridging full-precision and low-bit neural networks, 2019.

He, K., Zhang, X., Ren, S., and Sun, J. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.90.

He, Y., Lin, J., Liu, Z., Wang, H., Li, L.-J., and Han, S. Amc: Automl for model compression and acceleration on mobile devices. *Lecture Notes in Computer Science*, pp. 815–832, 2018.

Hernández-Lobato, J. M., Gelbart, M. A., Reagen, B., Adolf, R., Hernández-Lobato, D., Whatmough, P. N., Brooks, D., Wei, G.-Y., and Adams, R. P. Designing neural network hardware accelerators with decoupled objective evaluations. In *NIPS workshop on Bayesian Optimization*, 2016.

Highlander, T. and Rodriguez, A. Very efficient training of convolutional neural networks using fast fourier transform and overlap-and-add. *Procedings of the British Machine Vision Conference 2015*, 2015. doi: 10.5244/c.29.160.

Horowitz, M. 1.1 computing's energy problem (and what we can do about it). In *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 10–14, Feb 2014. doi: 10.1109/ISSCC.2014.6757323.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q. V., and Adam, H. Searching for mobilenetv3, 2019.

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, 2017.

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., and Keutzer, K. Squeezenet: Alexnet-level accuracy with 50x fewer parameters and ¡0.5mb model size, 2016.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., Adam, H., and Kalenichenko, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00286.

Kalamkar, D., Mudigere, D., Mellempudi, N., Das, D., Banerjee, K., Avancha, S., Vooturi, D. T., Jammalamadaka, N., Huang, J., Yuen, H., Yang, J., Park, J., Heinecke, A., Georganas, E., Srinivasan, S., Kundu, A., Smelyanskiy, M., Kaul, B., and Dubey, P. A study of bfloat16 for deep learning training, 2019.

Kingma, D. P. and Ba, J. Adam: A method for stochastic optimization. In *International Conference on Learning Representations (ICLR)*, 2015.

Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. *CoRR*, abs/1806.08342, 2018.

Krizhevsky, A. et al. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

L. Toom, A. The complexity of a scheme of functional elements realizing the multiplication of integers. *Doklady Akademii Nauk SSSR*, 3, 01 1963.

Lane, N. D. and Warden, P. The deep (learning) transformation of mobile and embedded computing. *Computer*, 51(5):12–16, May 2018. ISSN 0018-9162. doi: 10.1109/MC.2018.2381129.

Lavin, A. and Gray, S. Fast algorithms for convolutional neural networks. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016. doi: 10.1109/cvpr.2016.435.

Lecun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pp. 2278–2324, 1998.

Lee, R., Venieris, S. I., Dudziak, L., Bhattacharya, S., and Lane, N. D. Mobisr: Efficient on-device super-resolution through heterogeneous mobile processors. In *The 25th Annual International Conference on Mobile Computing and Networking*, MobiCom '19, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450361699. doi: 10.1145/3300061.3345455.

Li, F., Zhang, B., and Liu, B. Ternary weight networks, 2016.

Li, H., Bhargav, M., Whatmough, P. N., and Philip Wong, H. . On-Chip Memory Technology Design Space Explorations for Mobile Deep Neural Network Accelerators. In *2019 56th ACM/IEEE Design Automation Conference (DAC)*, pp. 1–6, June 2019.

Liberis, E. and Lane, N. D. Neural networks on microcontrollers: saving memory at inference via operator reordering, 2019.

Lin, X., Zhao, C., and Pan, W. Towards accurate binary convolutional neural network. In *Advances in Neural Information Processing Systems*, pp. 345–353, 2017.

Liu, H., Simonyan, K., and Yang, Y. DARTS: Differentiable architecture search. In *International Conference on Learning Representations*, 2019.

Liu, X., Pool, J., Han, S., and Dally, W. J. Efficient sparse-winograd convolutional neural networks. In *International Conference on Learning Representations*, 2018.

Loshchilov, I. and Hutter, F. Sgdr: Stochastic gradient descent with warm restarts. In *International Conference on Learning Representations (ICLR) 2017 Conference Track*, April 2017.

Maji, P., Mundy, A., Dasika, G., Beu, J., Mattina, M., and Mullins, R. Efficient winograd or cook-toom convolution kernel implementation on widely used mobile cpus, 2019.

Mathieu, M., Henaff, M., and LeCun, Y. Fast training of convolutional networks through ffts, 2013.

Meng, L. and Brothers, J. Efficient winograd convolution via integer arithmetic. *CoRR*, abs/1901.01965, 2019.

Paszke, A., Gross, S., Chintala, S., Chanan, G., Yang, E., DeVito, Z., Lin, Z., Desmaison, A., Antiga, L., and Lerer, A. Automatic differentiation in PyTorch. In *NIPS Autodiff Workshop*, 2017.

Rastegari, M., Ordonez, V., Redmon, J., and Farhadi, A. Xnor-net: Imagenet classification using binary convolutional neural networks. *CoRR*, abs/1603.05279, 2016.

Real, E., Aggarwal, A., Huang, Y., and Le, Q. V. Regularized evolution for image classifier architecture search. *Proceedings of the AAAI Conference on Artificial Intelligence*, 33(01):4780–4789, Jul. 2019. doi: 10.1609/aaai.v33i01.33014780.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., and Chen, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Jun 2018. doi: 10.1109/cvpr.2018.00474.

Simonyan, K. and Zisserman, A. Very deep convolutional networks for large-scale image recognition. In *International Conference on Learning Representations*, 2015.

Stamoulis, D., Ding, R., Wang, D., Lymberopoulos, D., Priyantha, B., Liu, J., and Marculescu, D. Single-path mobile automl: Efficient convnet design and nas hyperparameter optimization, 2019.

Strassen, V. Gaussian elimination is not optimal. *Numer. Math.*, 13(4):354–356, August 1969. ISSN 0029-599X. doi: 10.1007/BF02165411.

Sutskever, I., Martens, J., Dahl, G., and Hinton, G. On the importance of initialization and momentum in deep learning. In Dasgupta, S. and McAllester, D. (eds.), *Proceedings of the 30th International Conference on Machine Learning*, volume 28 of *Proceedings of Machine Learning Research*, pp. 1139–1147, Atlanta, Georgia, USA, 17–19 Jun 2013. PMLR.

Sze, V., Chen, Y.-H., Yang, T.-J., and Emer, J. S. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, Dec 2017. ISSN 1558-2256. doi: 10.1109/jproc.2017.2761740.

Takikawa, T., Acuna, D., Jampani, V., and Fidler, S. Gated-scnn: Gated shape cnns for semantic segmentation. *arXiv preprint arXiv:1907.05740*, 2019.

Tan, M. and Le, Q. V. Efficientnet: Rethinking model scaling for convolutional neural networks, 2019.

Tan, M., Chen, B., Pang, R., Vasudevan, V., and Le, Q. V. Mnasnet: Platform-aware neural architecture search for mobile. In *CVPR*, 2019.

Tschannen, M., Khanna, A., and Anandkumar, A. StrassenNets: Deep learning with a multiplication budget. In Dy, J. and Krause, A. (eds.), *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pp. 4985–4994, Stockholmsmässan, Stockholm Sweden, 10–15 Jul 2018. PMLR.

Vincent, K., Stephano, K., Frumkin, M., Ginsburg, B., and J., D. On improving the numerical stability of winograd convolutions. In *International Conference on Learning Representations (Workshop track)*, 2017.

Wan, D., Shen, F., Liu, L., Shao, L., Qin, J., and Shen, H. T. Tbn: Convolutional neural network with ternary inputs and binary weights. In *ECCV*, 2018.

Wang, K., Liu, Z., Lin, Y., Lin, J., and Han, S. Haq: Hardware-aware automated quantization with mixed precision. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

Whatmough, P. N., Lee, S. K., Brooks, D., and Wei, G. DNN Engine: A 28-nm Timing-Error Tolerant Sparse Deep Neural Network Processor for IoT Applications. *IEEE Journal of Solid-State Circuits*, 53(9):2722–2731, Sep. 2018. ISSN 1558-173X. doi: 10.1109/JSSC.2018.2841824.

Whatmough, P. N., Lee, S. K., Donato, M., Hsueh, H., Xi, S., Gupta, U., Pentecost, L., Ko, G. G., Brooks, D., and Wei, G. A 16nm 25mm2 SoC with a 54.5x Flexibility-Efficiency Range from Dual-Core Arm Cortex-A53 to eFPGA and Cache-Coherent Accelerators. In *2019 Symposium on VLSI Circuits*, pp. C34–C35, June 2019. doi: 10.23919/VLSIC.2019.8778002.

Whatmough, P. N., Zhou, C., Hansen, P., Venkataramanaiah, S. K., Seo, J.-S., and Mattina, M. FixyNN: Efficient Hardware for Mobile Computer Vision via Transfer Learning, 2019.

Winograd, S. *Arithmetic Complexity of Computations*. Society for Industrial and Applied Mathematics, 1980. doi: 10.1137/1.9781611970364.

Winograd, S. Signal processing and complexity of computation. In *ICASSP '80. IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 5, pp. 94–101, April 1980. doi: 10.1109/ICASSP.1980.1171044.

Xiang, X., Qian, Y., and Yu, K. Binary deep neural networks for speech recognition. In *Proc. Interspeech 2017*, pp. 533–537, 2017. doi: 10.21437/Interspeech.2017-1343.

Xie, S., Girshick, R. B., Dollár, P., Tu, Z., and He, K. Aggregated residual transformations for deep neural networks. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 5987–5995, 2017.

Zhang, Y., Li, K., Li, K., Wang, L., Zhong, B., and Fu, Y. Image super-resolution using very deep residual channel attention networks. *CoRR*, abs/1807.02758, 2018.

Zoph, B. and Le, Q. V. Neural architecture search with reinforcement learning. In *International Conference on Learning Representations*, 2017.

A APPENDIX

A.1 Winograd-aware layers for other architectures

The results of our study of Winograd-aware networks presented Section 6 showed multiple configurations of the ResNet-18 architecture at different width-multipliers, bit-widths, quantization levels and convolution algorithms. Here, we present a similar analysis for two other popular architectures for image classification. We limit our study to the full models (i.e. mult=1.0) We show results for SqueezeNet (Iandola et al., 2016) in Table 4 and for ResNeXt (Xie et al., 2017) in Table 5. These results align with what was observed for ResNet-18: In the presence of quantization, learning the Winograd transformations (*flex* configurations) resulted in superior performance than using the default (*static*) transformations. All experiments used the same hyper-parameters as described in Section 5.

Conv. Type	Bits act. / param.	WA trans.	Accuracy (%) CIFAR-10	CIFAR-100
im2row		-	91.13	69.06
WA_{F2}		static	91.31	69.42
WA_{F2}	32 / 32	flex	91.25	69.36
WA_{F4}		static	91.23	69.14
WA_{F4}		flex	91.41	69.32
im2row		-	91.15	69.34
WA_{F2}		static	90.88	70.06
WA_{F2}	8 / 8	flex	91.03	70.18
WA_{F4}		static	79.28	55.84
WA_{F4}		flex	90.72	69.73

Table 4. Comparison between standard convolutions (*im2row*) and Winograd-aware layers for SqueezeNet. With INT8 quantization and using the default transformation matrices (*static*), larger tile sizes (i.e. F4) introduce substantial numerical error and result in a sever accuracy drop. This drop in accuracy is significantly reduced if the transformations are learnt (*flex*).

For both architectures, INT8 Winograd-aware $F4$ models with learnt Winograd transformations did not result in a accuracy gaps as pronounced as the ones reported for ResNet-18 in Section 6. These models even surpass the *im2row* baselines for CIFAR-100. We argue this is because SqueezeNet and ResNeXt_(8×16) have fewer 3×3 convolutional layers (8 and 6, respecitvely) compared to ResNet-18, which has 16. Therefore, the succession of fewer convolutional layers implemented as Winograd convolutions reduces the overall impact of numerical error.

A.2 Overhead of Learnt Winograd Transforms

The default Winograd transformation matrices contain varying amounts of 0's. For $F2$ the sparsity ratios are 50%, 33% and 25% respectively for B^T, G and A^T. From the construction process of these matrices and specially the choice of *polynomial points*, we would expect lower sparsity ratios as these transforms are adjusted for larger input patches. For example, for the default transforms $F4$ these ratios are 22%, 22% and 25%. For implementations of matrix-matrix multiplications that can exploit data sparsity, as is the case of Arm's Compute Library, having zeros means less compute which often translate into lower latencies.

Conv. type	Bits act. / param.	WA trans.	Accuracy (%) CIFAR-10	CIFAR-100
im2row		-	93.17	74.54
WA_{F2}		static	93.19	74.66
WA_{F2}	32 / 32	flex	93.08	74.58
WA_{F4}		static	93.24	74.47
WA_{F4}		flex	93.15	74.62
im2row		-	93.40	74.89
WA_{F2}		static	92.93	75.32
WA_{F2}	8 / 8	flex	93.11	75.80
WA_{F4}		static	76.73	51.20
WA_{F4}		flex	93.29	75.35

Table 5. Comparison between standard convolutions (*im2row*) and Winograd-aware layers for ResNeXt_20(8×16). With INT8 quantization and using the default transformation matrices (*static*), larger tile sizes (F4) introduce substantial numerical error and result in a sever accuracy drop. This drop in accuracy is significantly reduced if the transformation matrices are learnt (*flex*).

The Winograd-aware formulation here presented doesn't impose restrictions on how the learnt transform should look like. As a consequence, the resulting transforms rarely contain zeros. This translates in additional compute for input $B^T dB$ and output $A^T yA$ transforms. The impact of using dense, learnt, transforms for WA_{F4} models running on a Cortex-A73 is a latency increase of 17% (+8ms) and 20% (+6ms) for FP32 and INT8 respectively for a ResNet18 network. This increase in latency is higher on the Cortex-A53 since the Winograd transforms are proportionally more expensive on this core. These penalties represent the worst case performance increase, assuming the transforms are compute bound. However, we believe that due to the access patterns of the Winograd transform kernels (gather and scatter across a wide area of memory) at least some of the performance of the transforms results from misses in the cache hierarchy and so some additional computation can be tolerated without necessarily increasing execution time.

We note that the impact for $F2$ models is considerably higher especially since the original transforms G and A are, not only sparse, but binary and the learnt ones are not. However, these penalties are never met in practice since $F2$ Winograd-aware models with default transforms can perform equally well as those with learnt transforms (as shown in Figure 4 and Tables 4 and 5) even in INT8.

Even with the performance loss due to the learnt transforms, we're still demonstrating some (non-negligible) $1.54\times$ and

$1.43\times$ speedup compared to INT8 *im2row* for A73 and A53 respectively. To the best of our knowledge this is the first time INT8 Winograd convolutions are empirically proven to work.

A.3 Architectures optimized with wiNAS

Our framework wiNAS, takes a given macro-architecture and optimizes each 3×3 convolutional layer by choosing from direct convolution or different Winograd configurations. For the search, all 1×1 convolutions were fixed to use *im2row*.

For wiNAS_{WA} in FP32, the resulting architecture only substituted the last convolution layer with *im2row* instead of $F2$. The rest of the layers remained unchanged from the WA_{F4} configuration (which was described in Section 5.1). The same micro-architecture was used in CIFAR-10 and CIFAR-100.

For wiNAS_{WA} with 8-bit quantization and CIFAR-10, wiNAS replaced the 5^{th} and second last convolutional layers with *im2row*, instead of $F4$ and $F2$ respectively. For CIFAR-100, more optimization was compared to WA_{F4}. The resulting micro-architecture optimization is shown in Figure 9 (left).

When introducing quantization in the search space, $\text{wiNAS}_{\text{WA-Q}}$, the resulting architectures are shown in Figure 9 for both CIFAR-10 (middle) and CIFAR-100 (right).

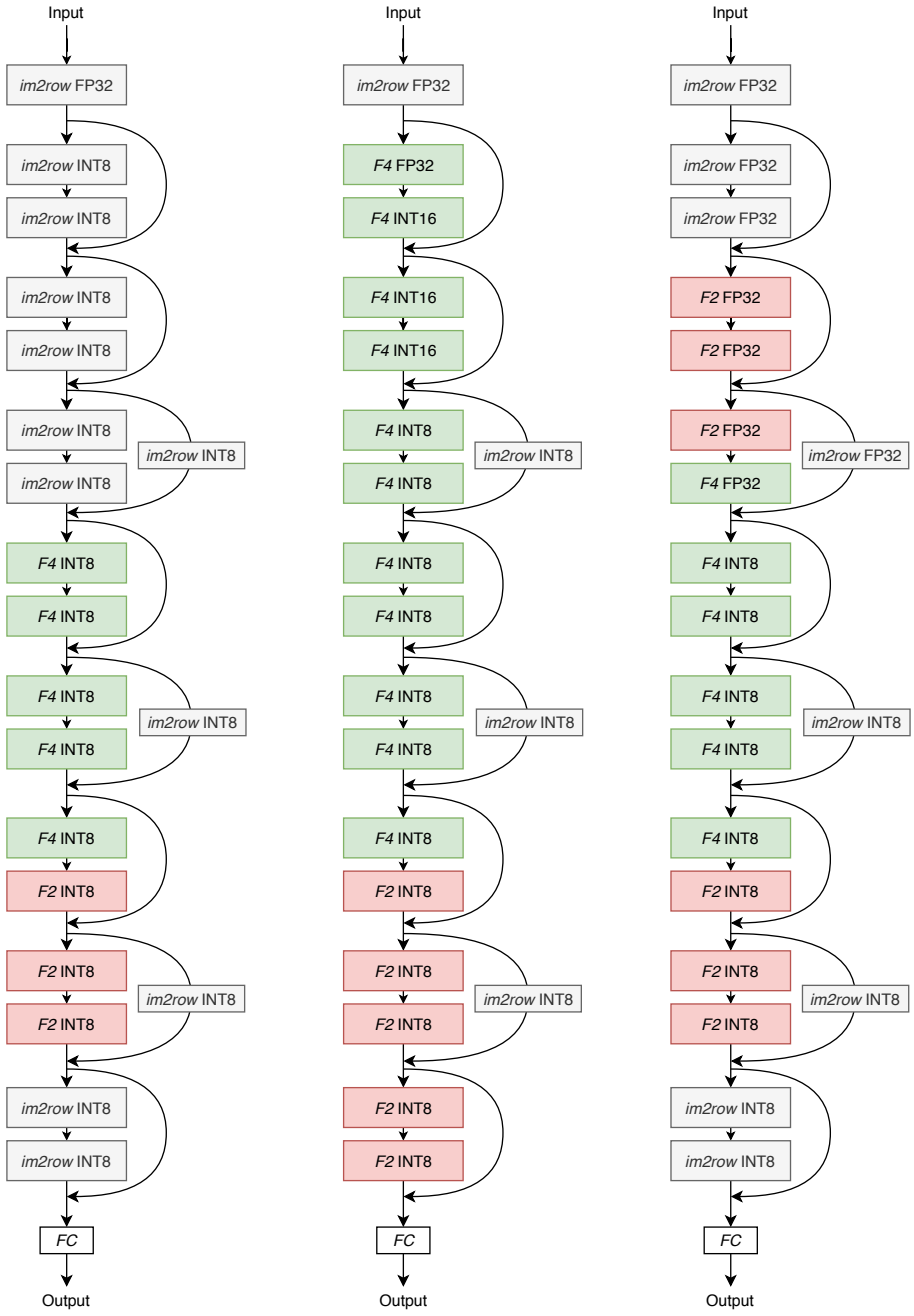

Figure 9. Resulting architectures after optimizing a ResNet-18 macro-architecture using wiNAS. For wiNAS$_{\text{WA}}$ and CIFAR-100, the architecture resulted is shown on the left. With wiNAS$_{\text{WA-Q}}$, that introduces quantization in the search space, the optimization resulted in different architectures for CIFAR-10 (middle) and CIFAR-100 (right), evidencing the difference in complexity of the latter.