Quantizing Deep Convolutional Networks for Efficient Inference

Raghu Krishnamoorthi, Facebook

Acknowledgements

- Most of the results presented here are from work done at Google as part of the Tensorflow lite team.
- Refer to work from several colleagues at google including: Benoit Jacob, Skiramantas Kligys, Dmitry Kalachenko, Suharsh Sivakumar and Pete Warden.
- We also refer to work from facebook on quantization, based on the work of Jongsoo Park, Bichen Wu, Maxim Naumov, Summer Deng, Marat Dukhan, Bichen Wu, Peizhao Zhang and Peter Vajda.

Outline

- Motivation
- Quantization: Overview
- Quantizing deep networks
  - Post Training quantization
  - Quantization aware training
- Lower precision inference
- Hardware accelerator recommendations
- Model system co-design
- Looking ahead

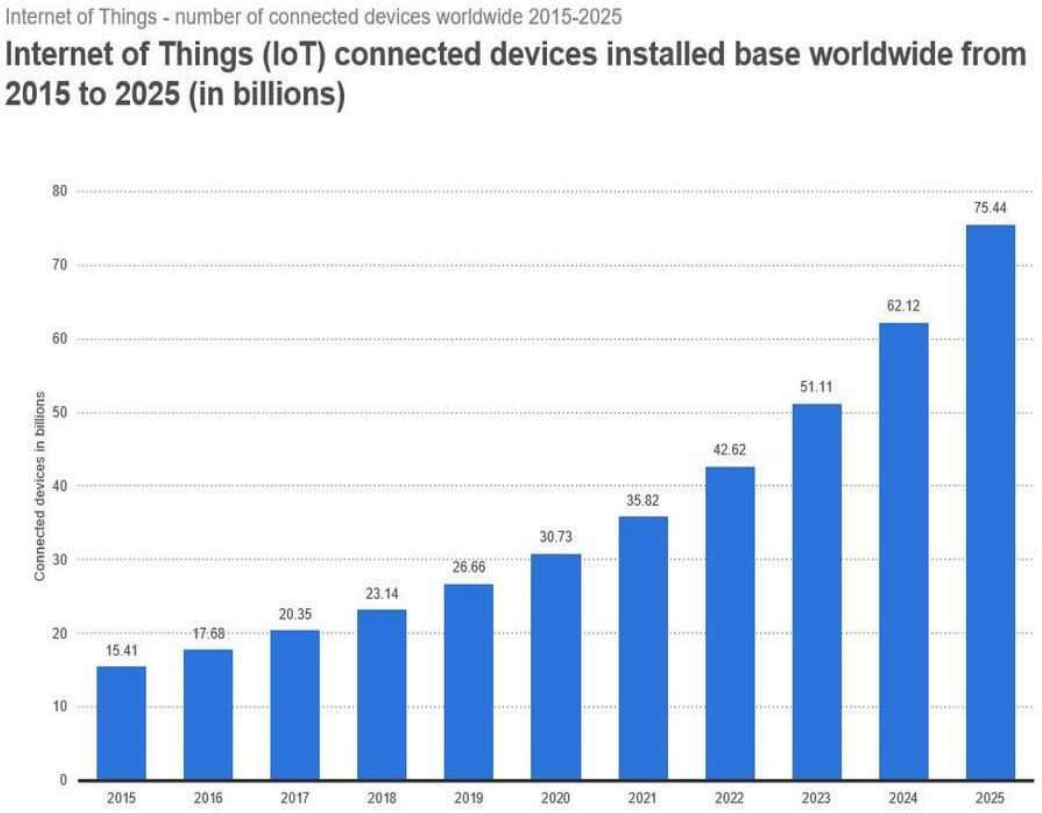
Motivation(1)

- Data-center power consumption is doubling every year

Source: Deep Learning Inference in Facebook Data-Centers [1]

Motivation(2)

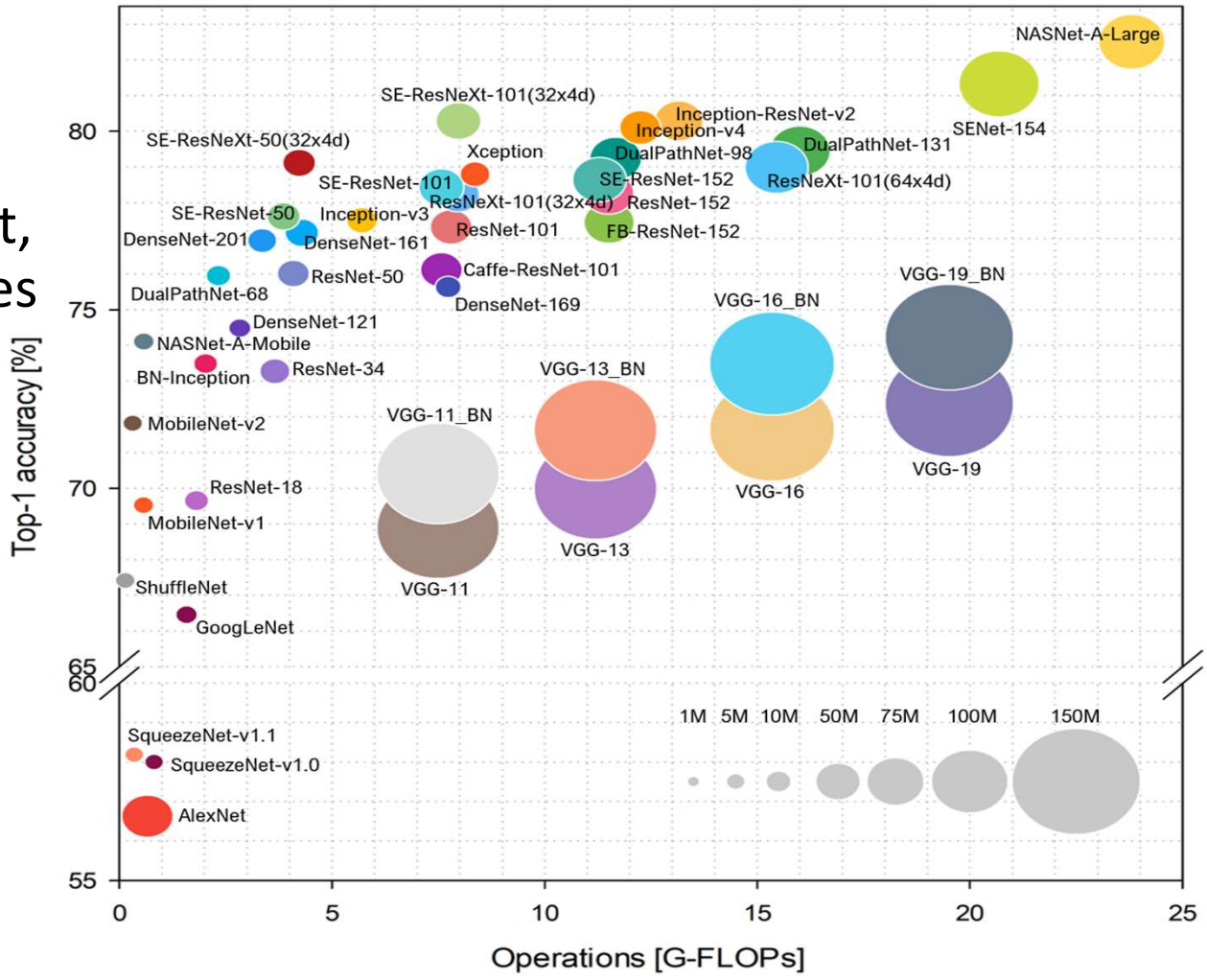
- Number of edge devices is growing rapidly, lots of these devices are resource constrained.



Source: https://www.statista.com/statistics/471264/iot-number-of-connected-devices-worldwide/

Motivation(3)

- While models are becoming more efficient, high accuracy still implies high complexity



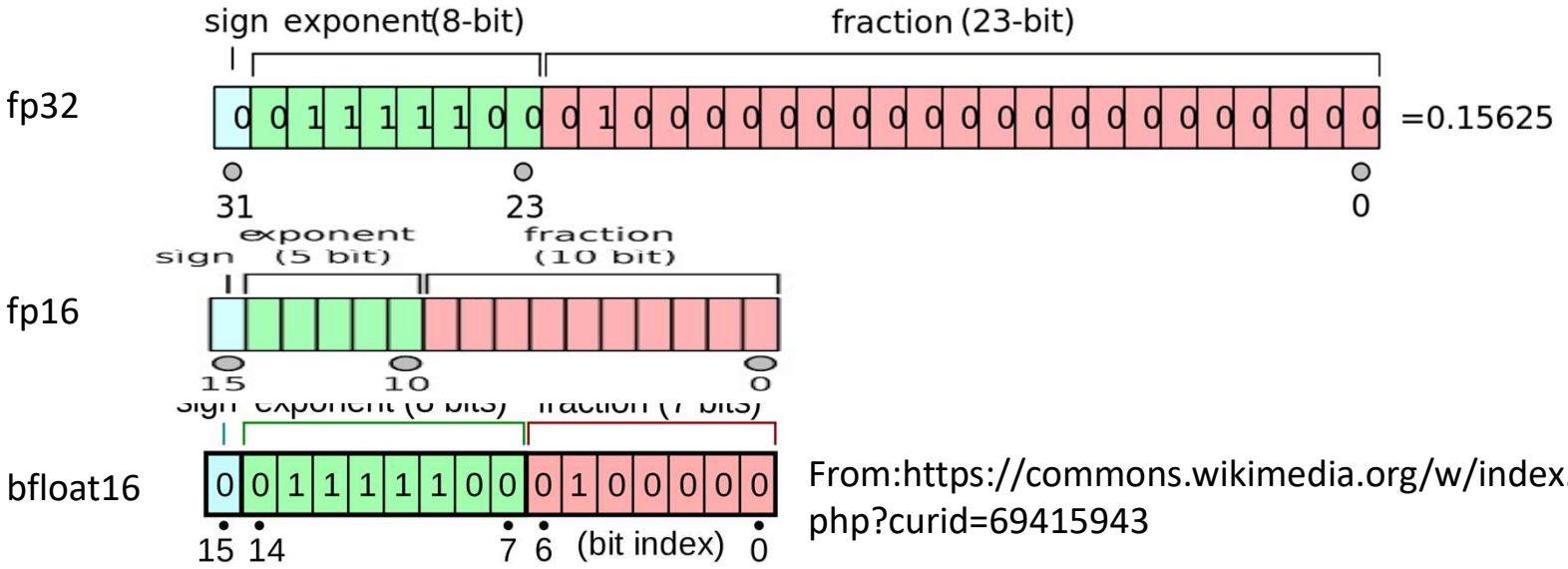
From: [Benchmark Analysis of Representative Deep Neural Network Architectures](), **Simone Bianco et al,**

Quantization

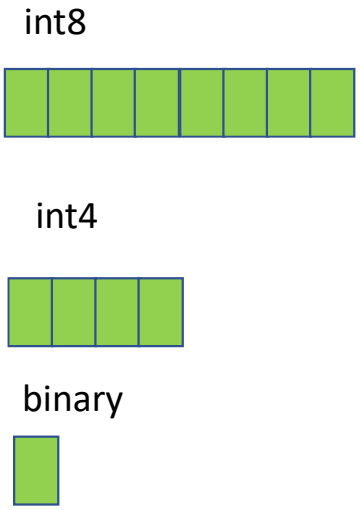
- Many approaches to solve the problems outlined here:
  - Better hardware accelerators: TPUs
  - Optimized kernels: Cudnn, Intel MKL-DNN, FBGemm,Nnpack,
  - Efficient deep network architectures: Nasnet, Mobilenet, FBNet
- A simpler approach that does not require re-design of models/new hardware is quantization.
  - Quantization refers to techniques to perform computation and storage at reduced precision

Background: Quantization(1)

- Quantization refers to mapping values from fp32 to a lower precision format.
  - Specified by
    - Format
    - Mapping type
    - Granularity

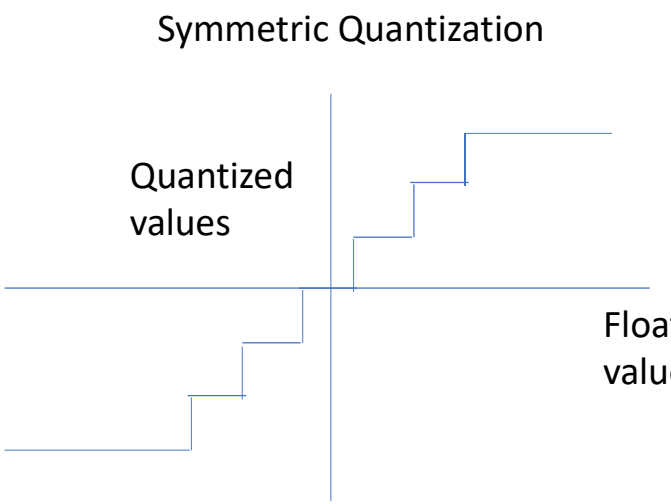


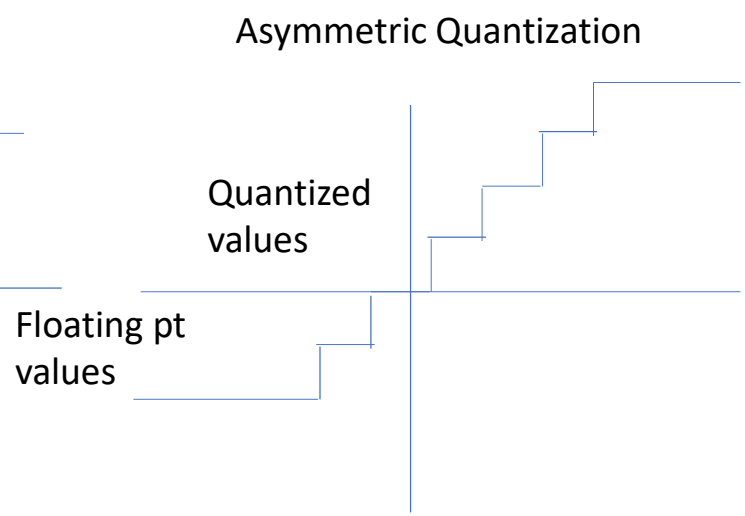
From:https://commons.wikimedia.org/w/index.php?curid=69415943

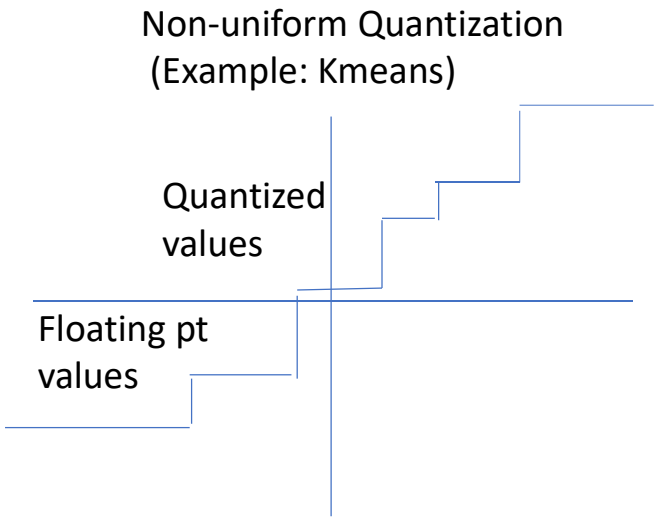


Background:Quantization(2)

- Mapping type
  - Uniform: Quantized values are uniformly spaced. Further divided into
    - Asymmetric: Quantized values obtained by rounding an affine transform of a floating point value
    - Symmetric: Quantized values are obtained by rounding a scaled floating point value
  - Non-uniform:
    - Quantized values are non-uniformly spaced

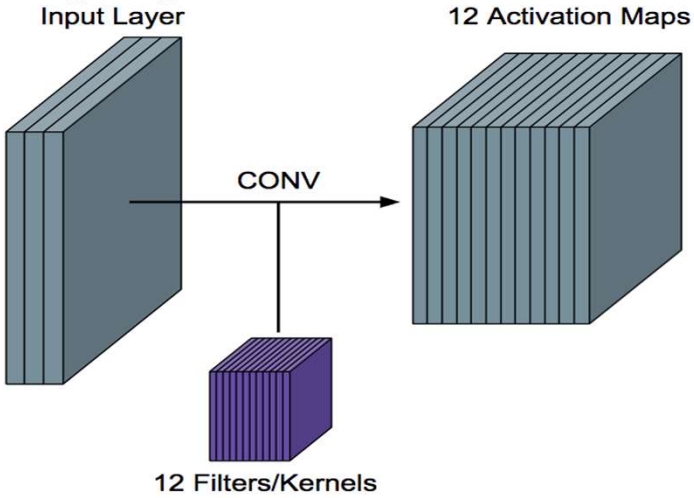


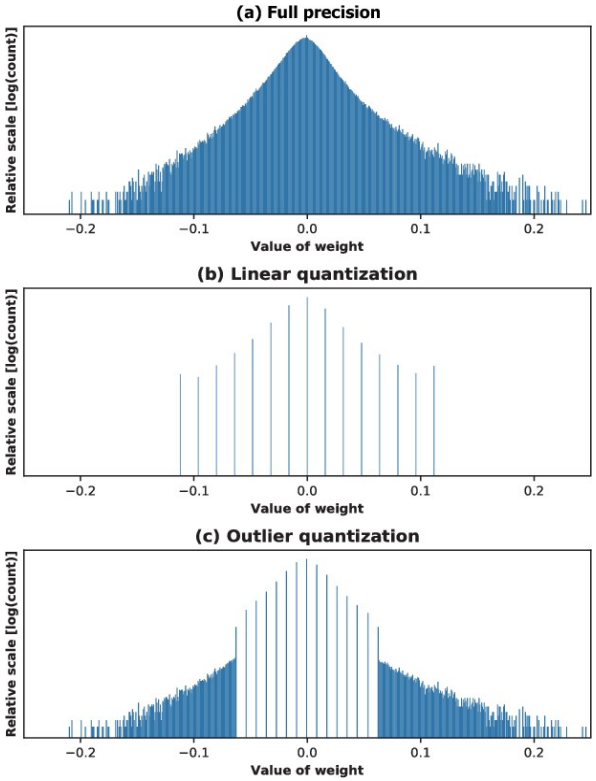




Background:Quantization(3)

- We also consider different granularities of quantization:
  - Per layer quantization
    - Same mapping for all elements in a layer.
  - Per row/ Per-channel quantization:
    - Choose quantizer parameters independently for each row (fc layers) or for each conv kernel (conv layers)
  - Outlier aware quantization:
    - Separate outliers to use lower precision arithmetic for bulk of weights.
    - Dense computations for inliers with sparse computation for outliers that have a large magnitude





Background: Quantization (4)

- So far, quantization is deterministic
- During training, we also consider modeling quantization stochastically:
  - For the case of symmetric quantization we perform stochastic quantization as:

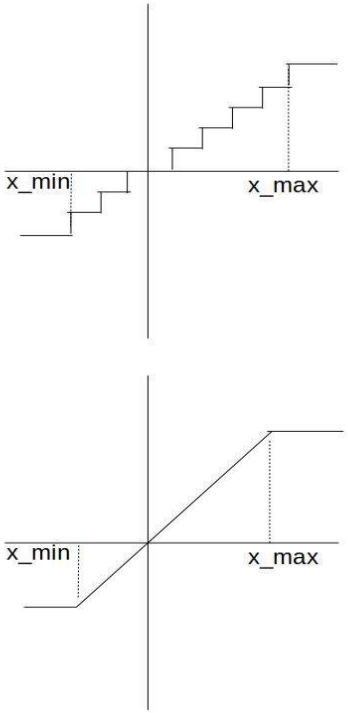
    $Q(x) = clamp\left(round\left(\frac{x+w}{s}\right)\right)$, where w is uniformly distributed in $(-\frac{s}{2}, \frac{s}{2})$ and s is the step size.
  - Stochastic quantization is equivalent to adding noise and has a regularizing effect [3]

Fake-quantization

- Emulate quantization by quantizing and de quantizing in succession
  - Values are still in floating point, but with reduc precision
- $x_{out} = FakeQuant(x)$

$$= s.\left(Clamp\left(round\left(\frac{x}{s}\right) - z\right) + z\right)$$

$$= DeQuant(Quant(x))$$



Fake Quantizer (top), showing the quantization of output values. Approximation for purposes of derivative calculation (bottom).

Quantization:Benefits

Benefits	Quantization
Applicability	Broad applicability across models and use cases
Support	Supported by x86, Nvidia Volta, ARM, Mali, QDSP
Software Support	Kernel libraries widely available
Memory Size	4x reduction
Memory Bandwidth/Cache	4x reduction
Compute	2x to 4x speedup
Power	Proportional to memory bandwidth savings

Quantization: Hardware Platforms

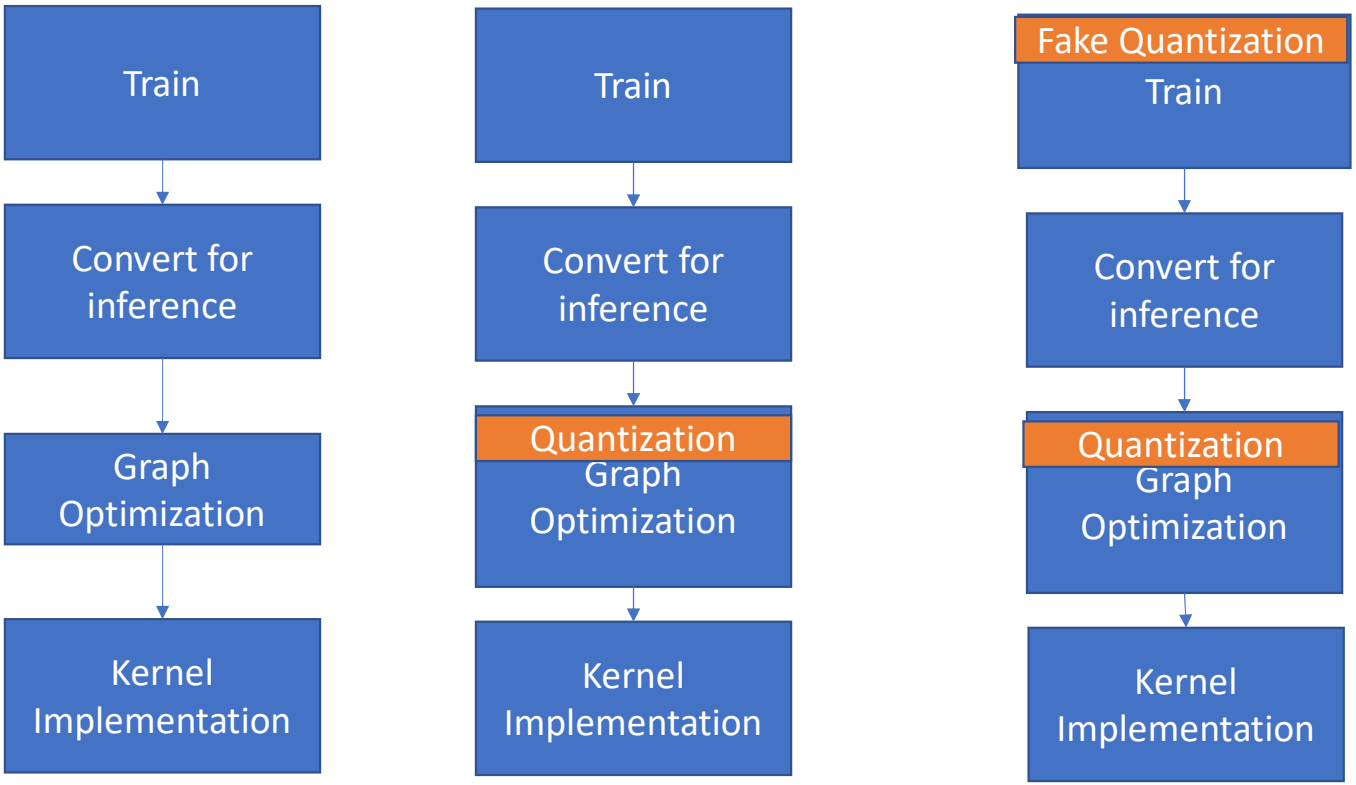
Platform	Precision Support	Kernel Library Support
ARM CPU	Fp16, int8	ARM-CMSIS, GEMM-LOWPE, QNNPACK
x86	Fp16, int8	MKL-DNN, FBGEMM
Nvidia Volta	Fp16, int8, int4 (Turing)	Cudnn, tensorRT
AMD Vega	Fp16, int8	ROCm
Qualcomm DSP	int8	SNPE, NN-API
ARM Mali	Fp16, int8	ARM NN SDK/NN-API

Quantization: Challenges

Challenges	Notes	Mitigation
Accuracy drop	Loss in accuracy can be too high for certain applications	Quantization aware training
Kernel Support	Wide variety of operators+multiple hardware platforms	Improving software tool-chain (TVM) to handle varied backends.
“Complexity”	Non-trivial: Requires calibration/training in some cases	Support in software packages: TensorRT, Tensorflow and Pytorch is improving

Quantizing deep networks

Model Quantization: Overview



What to quantize?

- Only quantize parts of network that contribute significantly to performance
  - May need to further reduce based on accuracy impact.
- Multiple ways to quantize a network with different impact:

Quantization scheme	Memory bandwidth reduction (Weights)	Memory bandwidth reduction (Activations)	Compute Speedup	Notes
Weight only quantization to int8	4x	1x	1x	Suitable for embedding lookups
Dynamic quantization	4x	1x	2x	Suitable for fc layers with small batches
Static quantization (int32 accumulators)	4x	4x	2x	Suited for all layers, important for convolutions
Static quantization (int16 accumulators)	4x	4x	4x	Requires lower precision accumulators

Post training quantization: Weight compression

- Simplest quantization scheme is to compress the weights to lower precision
  - Requires no input data and can be done statically as part of preparing a model for inference
  - Hardware accelerators can benefit if de-compression is done after memory access
    - Trivial for case of fp16/int8 quantization of weights.
  - K-means compression is also supported in select platforms and is amenable to simple de-compression
    - Scatter-Gather operation in select processors
    - Supported in CoreML

Dynamic quantization

- Dynamic quantization refers to schemes where the activations are read/written in fp32 and are dynamically quantized to lower precisions for compute.
- Requires no calibration data
- Data exchanged between operations is in floating point, so no need to worry about format conversion.
- Provides performance improvements close to static quantization when memory access is dominated by weights
  - Suitable for inference in RNNs
  - Lesser gains for conv layers
- Supported by:
  - DNNLOWP (facebook)
  - Tensorflow Lite (Google)

Quantizing weights and activations

- Post training quantization refers to quantizing both weights and activations to reduced precision, typically int8.
- Requires estimation of statistics of activations for determining quantizer parameters.
- Quantizer parameters are determined by minimizing an error metric:
  - KL Divergence: TensorRT
  - Saturation error: Tensorflow Lite

Results

Setup

- All results were generated using tensorflow with weights and activations being fake quantized to 8 bits unless stated otherwise.
- Top-1 accuracy is evaluated on 50000 validation images of Imagenet.
- Batch norm was folded into the convolution prior to quantization for inference (except for pre-activation pattern in Resnet-v2).
- Input image sizes were 224x224 or 299x299, matching the reference floating point implementations at: https://github.com/tensorflow/models/blob/master/research/slim/eval_image_classifier.py
- For weights we consider the following combinations:
  - Asymmetric Per layer
  - Symmetric per-layer
  - Asymmetric per-channel
  - Symmetric per-channel
- For activations, we consider only per-layer quantization
  - Since activations are unsigned, there is no distinction between symmetric/asymmetric quantization

Baseline

Accuracy metrics measured on Imagenet validation set of 50000 images

Network	Model Size (Num Parameters)	Top-1 Accuracy (Imagenet)
Mobilenet-v1-0.25-128	0.47M	0.415
Mobilenet-v2-1-224	3.54M	0.719
Mobilenet-v1-1-224	4.25M	0.709
Nasnet-Mobile	5.3M	0.74
Mobilenet-v2-1.4-224	6.06M	0.749
Inception-v3	23.9M	0.78
Resnet-v1-50	25.6M	0.752
Resnet-v2-50	25.6M	0.756
Resnet-v1-152	60.4M	0.768
Resnet-v2-152	60.4M	0.778

Network Architectures

- Primarily convolutions followed by batch normalization and Relu units
  - Kernels of different sizes, depth-wise, point-wise and group wise convolutions across networks
- Skip connections to improve accuracy in certain cases
- Batch normalization is used in all cases
  - Training: $y = \gamma\left(\frac{x-\mu_B}{\sigma_B}\right) + \beta$, μ_B , σ_B are batch mean and standard deviation
  - Inference $y = \gamma\left(\frac{x-\mu}{\sigma}\right) + \beta$, μ, σ are estimated mean and standard deviation from training
- Non-linearity: Either Relu or Relu6, which is a Relu which saturates activations beyond 6.

Weight only quantization: Results

Network	Asymmetric, Per Layer	Symmetric, Per Channel	Asymmetric, Per Channel	Floating point
Mobilenet-v2-1-224	0.001	0.698	0.698	0.719
Mobilenet-v1-1-224	0.001	0.591	0.704	0.709
Nasnet-Mobile	0.72	0.72	0.74	0.74
Mobilenet-v2-1.4-224	0.004	0.74	0.74	0.749
Inception-v3	0.78	0.78	0.78	0.78
Resnet-v1-50	0.75	0.75	0.75	0.752
Resnet-v2-50	0.75	0.75	0.75	0.756
Resnet-v1-152	0.766	0.763	0.762	0.768
Resnet-v2-152	0.761	0.76	0.77	0.778

Post training quantization: Results

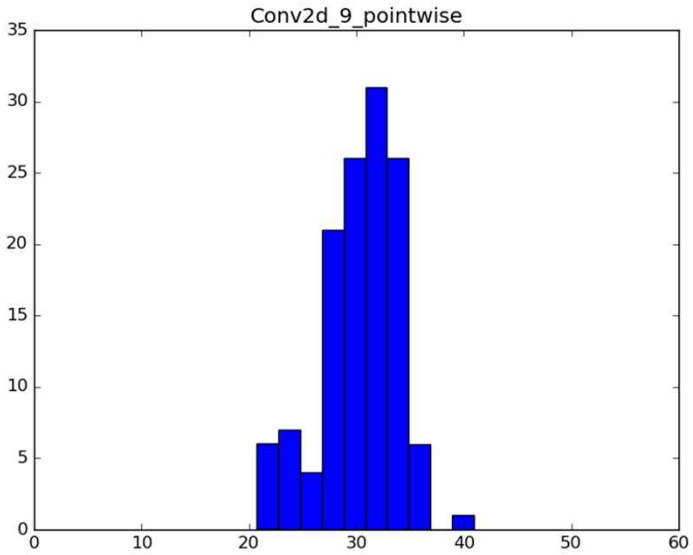
Network	Asymmetric, Per Layer	Symmetric, Per Channel	Asymmetric, Per Channel	Activation only quantized	Floating point
Mobilenet-v2-1-224	0.001	0.698	0.697	0.7	0.719
Mobilenet-v1-1-224	0.001	0.591	0.703	0.708	0.709
Nasnet-Mobile	0.72	0.72	0.74	0.74	0.74
Mobilenet-v2-1.4-224	0.004	0.74	0.74	0.742	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1-50	0.75	0.75	0.75	0.751	0.752
Resnet-v2-50	0.75	0.75	0.75	0.75	0.756
Resnet-v1-152	0.766	0.762	0.767	0.761	0.768
Resnet-v2-152	0.766	0.762	0.767	0.761	0.778

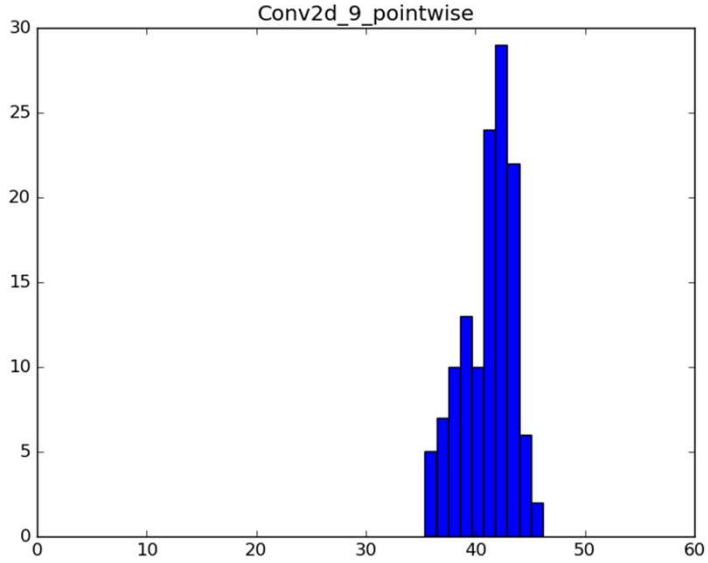
Observations

- Granularity of quantization:
  - Per-channel quantization can provide good accuracy and can be a good baseline for post training quantization of weights and activations, with asymmetric quantization providing close to floating point accuracy for all networks.
- Model Size:
  - Networks with more parameters like Resnets and Inception-v3 are more robust to quantization compared to Mobilenets which have fewer parameters.
  - There is a large drop when weights are quantized at the granularity of a layer, particularly for Mobilenet architectures.
- Weight vs activation quantization:
  - Activations can be quantized to 8-bits with almost no loss in accuracy.
  - The dynamic ranges of the activations are low due to a combination of:
    - Batch normalization with no scaling: Used in Inception-V3, which ensures that the activations of all feature maps have zero mean and unit variance.
    - ReLu6: Used in Mobilenets, which restricts the activations to be in a fixed range (0,6) for all feature maps, thereby removing large dynamic range variations.

Observations(2)

- Why are mobilenets so hard to quantize?
  - Batch normalization with strict limits on activation ranges (ReLu6) make the folded weights have a large dynamic range.

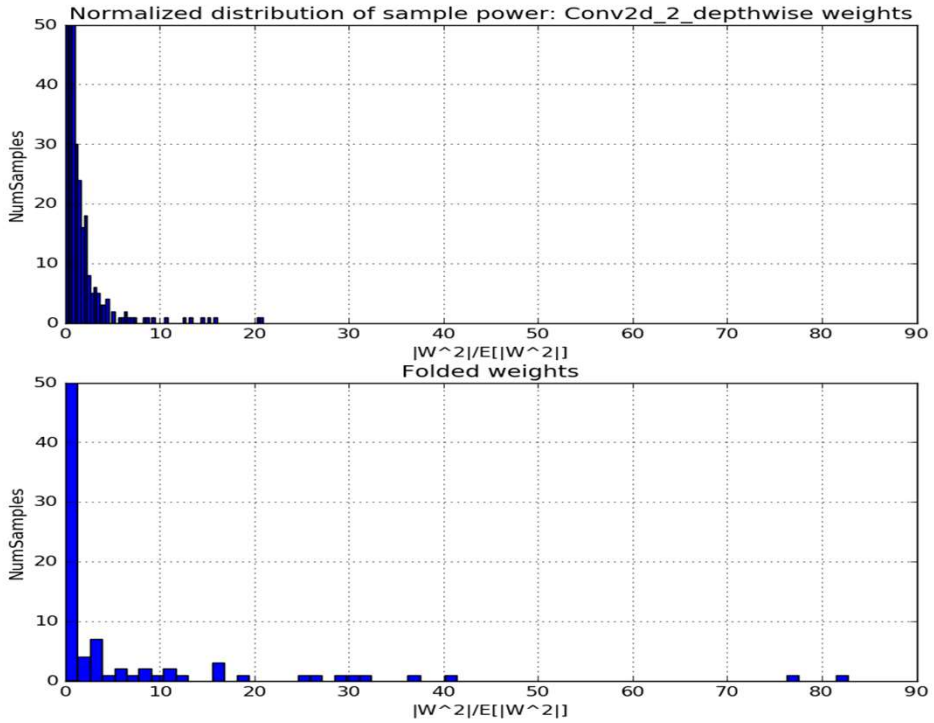




Histogram of SQNR per convolutional kernel: Asymmetric, per layer (left), symmetric, per-channel (right)

Observations (3)

- Note large outliers for folded weights

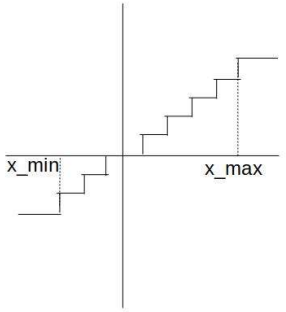


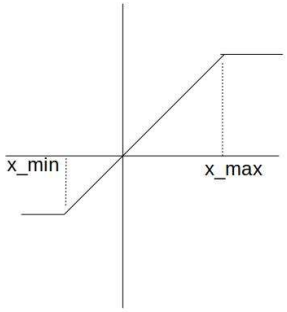
Quantization aware training

Quantization aware training(1)

- Model all quantization artifacts during training
  - Not to be confused with reduced precision training
- Fake-quantization to model quantization errors
  - For training, we need to model the backward pass. An approach that has worked well is the “straight through estimator” (See [5]). We model the quantizer as a piecewise linear function for purposes of calculating the derivative.
- Can also model quantization during training by stochastic quantization instead of deterministic rounding in the forward pass.

Forward pass

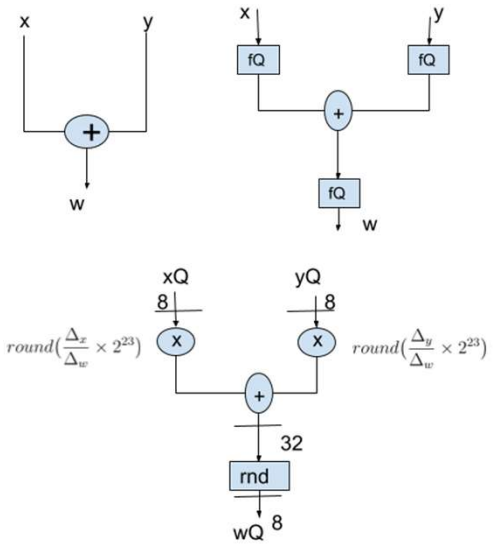


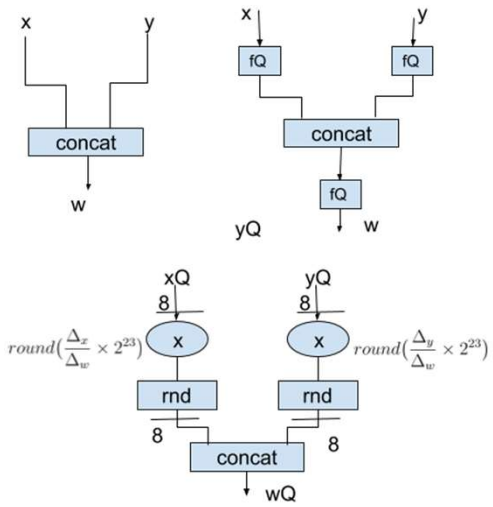


Backward pass

Quantization aware training(2)

- All quantization related artifacts need to be faithfully modeled at training time.
- Example: Add and Concat involve re-scaling when one models the actual fixed point implementation during training





Quantization Aware Training: Batch Normalization

- For quantization, we need to model behavior of batch norm at inference time during training.[4],[6]
  - During inference, batch norm is folded into the weights and biases

$$W_{Inf} = \frac{\gamma W}{\sigma}$$

$$Bias_{Inf} = \beta - \frac{\gamma \mu}{\sigma}$$

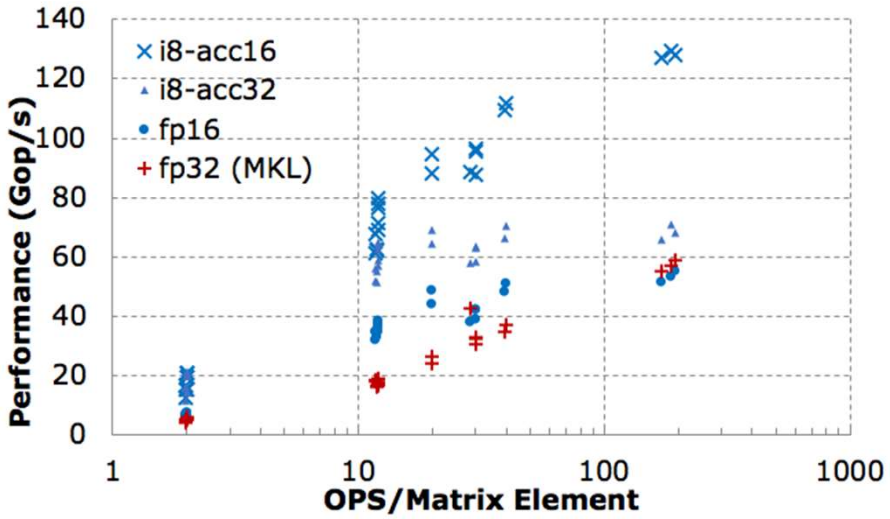
  - However, at training batch norm uses batch statistics.
- Non-trivial to model batch norm correctly during training, for more details see [6]
- Tensorflow provides helper functions to perform this as a graph rewrite.

Results

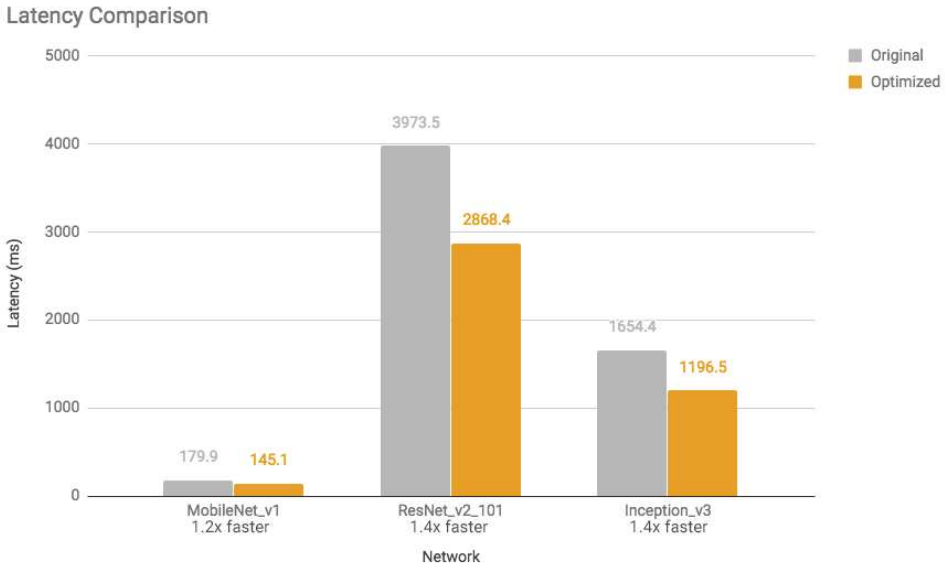
- Quantization aware training provides the best accuracy and allows for simpler quantization schemes.

Network	Asymmetric, Per Layer, post training quant	Symmetric, Per Channel, post training quant	Asymmetric, Per Layer, QAT	Symmetric, Per channel QAT	Floating point
Mobilenet-v2-1-224	0.001	0.698	0.709	0.711	0.719
Mobilenet-v1-1-224	0.001	0.591	0.70	0.707	0.709
Nasnet-Mobile	0.72	0.72	0.73	0.73	0.74
Mobilenet-v2-1.4-224	0.004	0.74	0.74	0.74	0.749
Inception-v3	0.78	0.78	0.78	0.78	0.78
Resnet-v1-50	0.75	0.75	0.75	0.751	0.752
Resnet-v2-50	0.75	0.75	0.75	0.75	0.756
Resnet-v1-152	0.766	0.762	0.765	0.762	0.768
Resnet-v2-152	0.766	0.762	0.76	0.76	0.778

Performance



Server: FBGEMM (quantized) vs MKL-DNN (fp32)



Mobile: Inference time: float vs quantized, TFLite, Pixel2
QNNPACK kernels provides an additional 2x speedup

Lower precision inference: Quantizing weights

- Four bit precisions for weights provides good accuracy, but needs to be done selectively.
  - Larger networks are more robust to lower precision
  - Quantization aware training is critical
  - Selectively quantizing layers of a network to different precisions can reduce the accuracy drop

Network \ QuantizationType	Asymmetric, per-layer (Post Training Quantization)	Symmetric,per-channel (Post Training Quantization)	Symmetric, per-channel (Quantization Aware Training)	Floating Point
Mobilenet_v1_1_224	0.02	0.001	0.65	0.709
Mobilenet_v2_1_224	0.001	0.001	0.62	0.719
Nasnet_Mobile	0.001	0.36	0.7	0.74
Mobilenet_v2_1.4_224	0.001	0.001	0.704	0.749
Inception-v3	0.5	0.71	0.76	0.78
Resnet_v1_50	0.002	0.54	0.732	0.752
Resnet_v1_152	0.001	0.64	0.725	0.768
Resnet_v2_50	0.18	0.72	0.73	0.756
Resnet_v2_152	0.18	0.74	0.74	0.778

4-bit weights, 8 bit activations: Top-1 accuracy results

Lower precision inference: Quantizing activations

- Activations can also be quantized to lower precision, but seem to be more sensitive to quantization
  - Weight quantization is a deterministic distortion that can be compensated during training
  - Activation quantization can only be partially compensated, as one can only exploit statistics of the activations. Individual inputs have some randomness that cannot be compensated for.
  - Weights can be quantized at a fine granularity (per-channel) without impacting performance. Quantizing activations at a finer granularity requires special hardware

Network \ QuantizationType	Post Training Quantization (8,4)	Quantization Aware Training (8,4)	Quantization Aware Training (4,8)	Floating Point
Mobilenet_v1_1_224	0.48	0.64	0.65	0.709
Mobilenet_v2_1_224	0.07	0.58	0.62	0.719
Resnet_v1_50	0.36	0.58	0.732	0.752
Nasnet_Mobile	0.04	0.4	0.7	0.74
Inception_v3	0.59	0.74	0.76	0.78

Lower precision inference(3)

- Different layers of a neural network have different sensitivity to quantization errors
- Exciting work on Differentiable architecture search [7] for determining precision allocations across layers, showing excellent performance:

		DNAS (ours)			TTQ	ADMM
		Full	MA	ME	Full	2bit	3bit
ResNet18	Acc	71.03	71.21 (+0.18)	69.58 (-1.45)	69.6	66.6 (-3.0)	68.0 (-1.6)
	Comp	1.0	11.2	21.1	1.0	16.0	10.7
ResNet34	Acc	74.12	74.61 (+0.49)	73.37 (-0.75)	-		
	Comp	1.0	10.6	19.0			

Table 3: Mixed Precision Quantization for ResNet on ImageNet for model size compression. In the table, we abbreviate accuracy as "Acc" and compression as "Comp". "MA" denotes the most accurate model from architecture search and "ME" denotes the most efficient model.

		DNAS (ours)			PACT	DoReFA	QIP	GroupNet
		arch-1	arch-2	arch-3	W4A4	W4A4	W4A4	W1A2G5
ResNet18	Acc	71.01	70.64	68.65	69.2	68.1	69.3	67.6
	Full Acc	71.03	71.03	71.03	70.2	70.2	69.2	69.7
	Acc Δ	-0.02	-0.39	-2.38	-1.0	-2.1	+0.1	-2.1
	Comp	33.2	62.9	103.5	64	64	64	102.4
ResNet34	Acc	74.21	73.98	73.23	-			
	Full Acc	74.12	74.12	74.12				
	Acc Δ	+0.09	-0.14	-0.89				
	Comp	40.8	59.0	87.4				

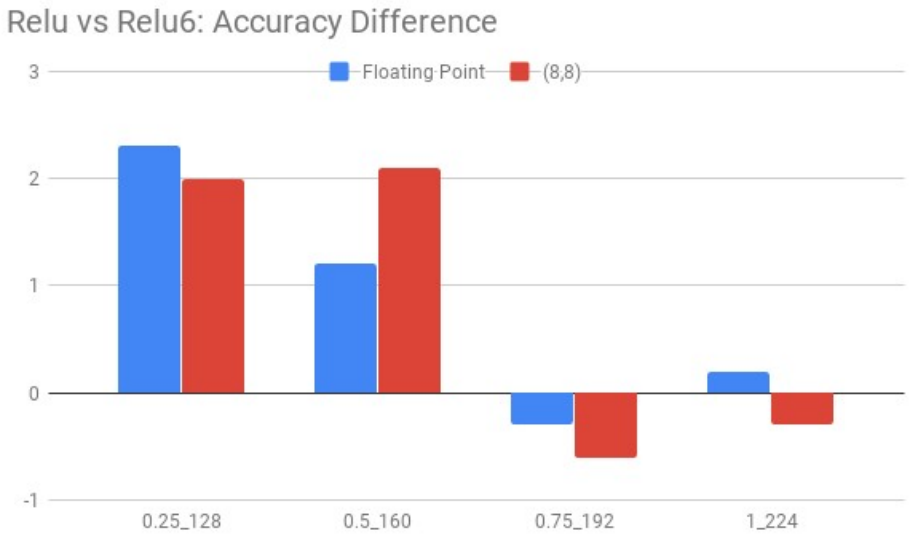
Table 4: Mixed Precision Quantization for ResNet on ImageNet for computational cost compression. We abbreviate accuracy as "Acc" and compression rate as "Comp". "arch-{1, 2, 3}" are three searched architectures ranked by accuracy.

Architecture trade-offs(1)

- Clear tradeoff between number of parameters and robustness to quantization
- One can also tradeoff number of feature maps vs precision
  - Having 2x number of feature maps at 4-bits is better than 8 bit quantization of the base network.

Architecture tradeoffs (2)

- Restricting ranges of activations apriori can hurt accuracy
  - Preferable to learn activation ranges instead of fixing them beforehand.



Co-design: Training quantized models

- Designing models that provide good quantization performance requires co-design of model architecture, training algorithms and hardware.
- Specific training enhancements include:
  - Fine tune from floating point models for building quantized models.
  - Freeze batch normalization statistics update to exactly model inference for further benefits.
  - Model exact rounding arithmetic done in hardware during training
    - Stochastic quantization provides models robust to random perturbations of weights, but underperforms techniques that model quantization as done at inference.
- Other enhancements to improve accuracy:
  - Use distillation to train quantized student from floating point teacher network [3]

Considerations for hardware accelerators

Reduce memory bandwidth

- Memory access can dominate power consumption
  - Compress weights and de-compress on the fly.
  - Consider exploiting sparsity of weights and activations for better compression.
- Don’t ignore activations: Most literature focusses on weights, activations can be very significant for large resolution inputs.
- Fuse multiple operations
  - Ensure hardware is flexible enough to fuse operations: Element-wise operations should always be done without requiring additional read/write to memory.

Support lower precision inference

- Consider supporting lower precisions for both weights and activations
  - For memory accesses, consider 2/4/6/8/16 bit precisions for weights and activations
  - For compute, consider 4/8 bit precisions, chosen independently for weights and activations
- Support quantization at a finer granularity
  - Per channel/Per row
  - Sparse engine for outlier aware quantization
- Symmetric quantization is sufficient, particularly with training.
- Consider supporting finer grained quantization for activations

Model hardware during training

- Provide support for modeling all hardware non-idealities during training:
  - Example: Using Relu vs Relu6 can affect training accuracy and exact hardware details matter
  - Example: Using 16-bit vs 32 bit accumulation should also be exposed during training.
  - Critical to get best performance/accuracy tradeoff.

Looking ahead (1)

- Co-design with hardware constraints
  - Meta learning approaches to jointly learn architecture and weights during training, taking device constraints into account, leading to efficient architectures.
- Example approaches include:
  - Differentiable architecture search:
    - B.Wu et al, Mixed precision quantization of convnets via differentiable neural architecture search
    - A.Gordon et al, Fast and Simple resource constrained structure learning of deep networks
  - Reinforcement Learning:
    - M.Tan et al, MnasNet: Platform-Aware Neural Architecture Search for Mobile
    - K. Wang et al, HAQ: Hardware-Aware Automated Quantization
  - Bayesian optimization:
    - X. Dai et al, ChamNet: Towards Efficient Network Design through Platform-Aware Model Adaptation
  - White-box approaches:
    - T. Yang, NetAdapt: Platform-Aware Neural Network Adaptation for Mobile Applications

Looking ahead(2)

- Ultra low precision networks for general use cases
  - Binary /ternary weights/activations with close to floating point accuracy for select networks (https://arxiv.org/abs/1612.01064v3, https://arxiv.org/abs/1612.01064v3)
- Novel approaches for quantization:
  - Quantization as regularizing loss function: M.Naumov et al, https://arxiv.org/pdf/1811.09862.pdf
  - Quantization as optimization: M.Carriera, https://arxiv.org/abs/1707.04319v1
- Quantization for the backward pass
  - Fp16 is already widely used: Nvidia, TPU (bfloat16)
  - Interesting work on using 8 bit precision for backprop: Scalable Methods for 8-bit Training of Neural Networks
- Quantization for distributed learning at the edge
  - Federated Learning: Support backward pass operations, but at a low duty cycle
  - Gradient compression
    - Similar techniques for compressing weights can be applied for gradients.

References

1. J Park, M. Naumov et al, “Deep Learning Inference in Facebook Data Centers: Characterization, Performance Optimizations and Hardware Implications”
2. Simone Bianco et al Benchmark Analysis of Representative Deep Neural Network Architectures
3. A. Polino et al, “Model compression via distillation and quantization”
4. B.Jacob, S.Kligys et al,``Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference''
5. M.Courbariaux, Y.Bengio et al, ``Binaryconnect: Training deep neural networks with binary weights during propagations’
6. R.Krishnamoorthi, “Quantizing deep convolutional networks for efficient inference: A whitepaper”
7. B.Wu et al, Mixed precision quantization of convnets via differentiable neural architecture search”