TensorFlow Lite

Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference

[Jacob *et al*. from Google 2017]

Ryo Takahashi

Motivation

Let’s get deeper into
optimized arithmetic
inside Neural Networks!!

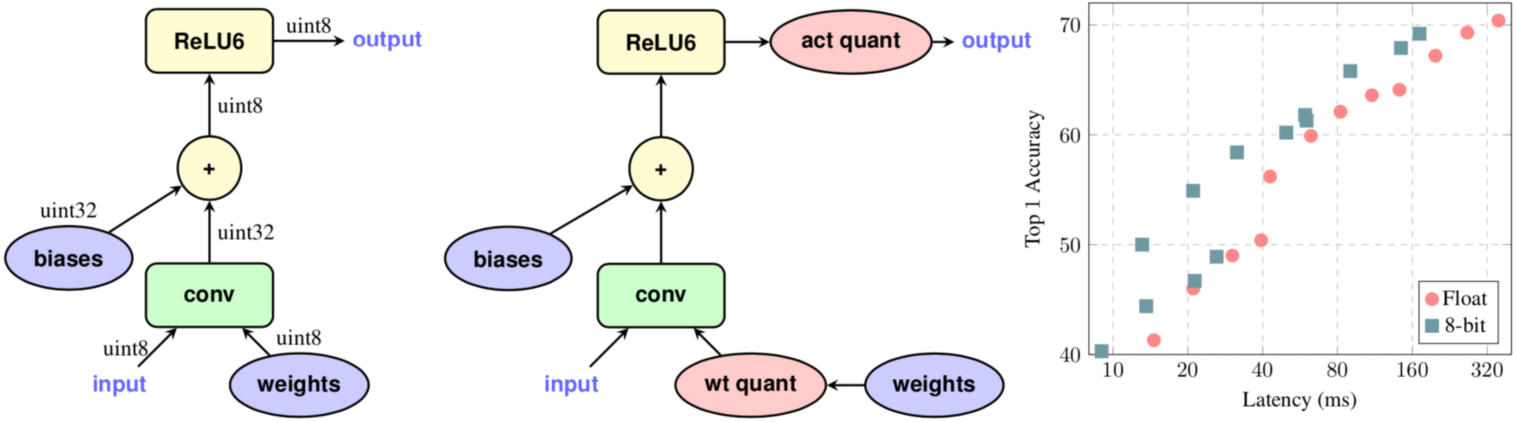
Approaches to CNN deployment on mobile platform

- Approach 1: computation/memory-efficient network architecture
  - e.g. MobileNet[arXiv:1704.04861], SqueezeNet[arXiv:1602.07360]
- Approach 2: quantization (Today's topic)
  - definition: quantize weights and activations from float into lower bit-depth format
  - benefit: save memory/power use, speed up inference

Existing works	Issues
• Ternary weight networks [arXiv:1605.04711] • Binary Neural networks [arXiv:1602.02505] $\underbrace{\begin{bmatrix} -0.4 & -0.4 & 0.9 \\ 0.9 & 0.4 & 0.8 \\ 0.4 & -0.4 & -0.4 \end{bmatrix}}_{W} \approx 0.2 \underbrace{\begin{bmatrix} -1 & -1 & 1 \\ 1 & 1 & 1 \\ 1 & -1 & -1 \end{bmatrix}}_{\alpha W^B}$ $W^B = \mathrm{sign}(W)$, $\alpha = \frac{1}{n} \lVert W \rVert_{\ell 1}$ these works can approximate conv. by bit-shifts/counts	• Their baseline architectures are over-parameterized - fat architectures (e.g. VGG) are easy to compress - it's still unclear that their schemes are applicable to modern light-weight architectures (e.g. MobileNet) - they are verified only in classification tasks, which are tolerant to quantization errors unlike regression • NOT efficient on common hardware (e.g. CPU) - bit-shifts/counts based conv. provides benefit only on custom hardware (e.g. FPGA, ASIC)

Proposal: Integer-arithmetic-only quantization

- improve latency-vs-accuracy tradeoffs of MobileNets on common hardware



(a) Integer-arithmetic-only inference (b) Training with simulated quantization (c) ImageNet latency-vs-accuracy tradeoff

a) Integer-arithmetic-only inference
- why convert weight and activation to not int8 but uint8 ?
- why keep the bit-depth of biases to 32bit?

b) Quantization-aware training
- quantize weight and activation during training unlike calibration

c) Evaluation in ImageNet classification and COCO object detection

OSS Contribution

- This work is included in Google's ML software stack:
  - TensorFlow (Model optimization)
  - TensorFlow Lite (Case studies)
  - Android NN

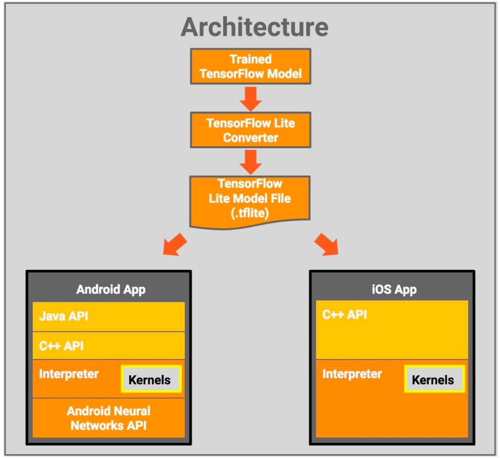
this work ↓

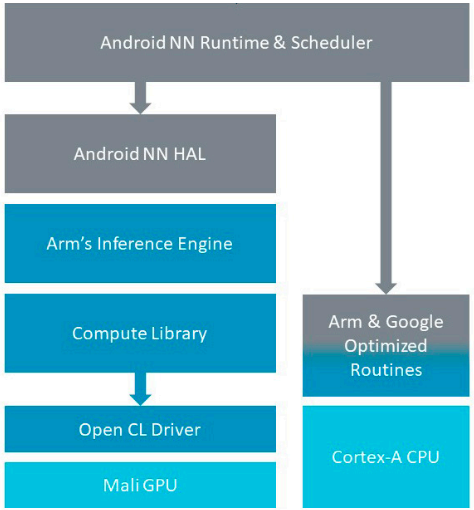
	Model	Top-1 Accuracy (Original)	Top-1 Accuracy (Post Training Quantized)	Top-1 Accuracy (Quantization Aware Training)	Latency (Original) (ms)	Latency (Post Training Quantized) (ms)	Latency (Quantization Aware Training) (ms)	Size (Original) (MB)	Size (Optimized) (MB)
light weight	Mobilenet-v1-1-224	0.709	0.657	0.70	124	112	64	16.9	4.3
light weight	Mobilenet-v2-1-224	0.719	0.637	0.709	89	98	54	14	3.6
fat	Inception_v3	0.78	0.772	0.775	1130	845	543	95.7	23.9
fat	Resnet_v2_101	0.770	0.768	N/A	3973	2868	N/A	178.3	44.9

big accuracy drop

small accuracy drop

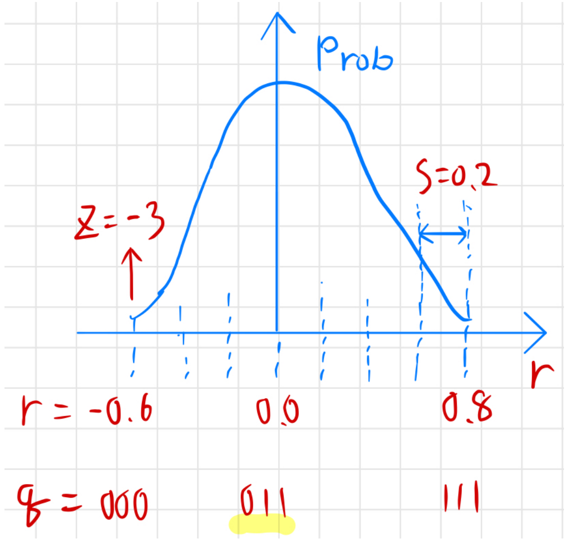
Table 1 Benefits of model quantization for select CNN models





Quantization scheme

- Equation: $r = S(q - Z)$
  - where:
    - r : real value
    - q : quantized value
    - S : scale (learned in training)
    - Z : zero-point (learned in training)
- Data structure in C++
  - create struct QuantizedBuffer for each weight and activation
  - each buffer has different S and Z



e.g. QType=uint3

```
template<typename QType>   // e.g. QType=uint8
struct QuantizedBuffer {
  vector<QType> q;         // the quantized values
  float S;                 // the scale
  QType Z;                 // the zero-point
};
```

Whey we can say integer-only-arithmetic in spite of this float S ?

Integer-arithmetic-only matrix multiplication

- Consider $X_3 = X_1 \cdot X_2$ where $X_\alpha = \begin{pmatrix} r_\alpha^{(0,0)} & \cdots & r_\alpha^{(0,N)} \\ \vdots & r_\alpha^{(i,j)} & \vdots \\ r_\alpha^{(N,0)} & \cdots & r_\alpha^{(N,N)} \end{pmatrix}$, $r_\alpha^{(i,j)} = S_\alpha(q_\alpha^{(i,j)} - Z_\alpha)$

$$S_3(q_3^{(i,k)} - Z_3) = \sum_{j=1}^{N} S_1(q_1^{(i,j)} - Z_1)S_2(q_2^{(j,k)} - Z_2),$$

which be rewritten as:

this $2N^3$ arithmetic operation stays in the inner loop

$$q_3^{(i,k)} = Z_3 + M\sum_{j=1}^{N}(q_1^{(i,j)} - Z_1)(q_2^{(j,k)} - Z_2) = Z_3 + M\left(NZ_1Z_2 - Z_1\boxed{\sum_{j=1}^{N} q_2^{(j,k)}} - Z_2\boxed{\sum_{j=1}^{N} q_1^{(i,j)}} + \boxed{\sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)}}\right)$$

these N^3 addition can be factored-out from calculation for each q_3

where:

$$M := \frac{S_1 S_2}{S_3} = 2^{-n} M_0$$

$\leftarrow$ M is empirically in (0,1)

where:

n is a non-negative integer

M_0 is a fixed-point value of typedef int32_t q31_t; // Q-format

Conv. & Affine get free from float-arithmetic by approximating M by int32_t

Implementation of a typical fused layer

$$\underbrace{Z_3 + 2^{-n} M_0}_{(2)} \underbrace{\left(N Z_1 Z_2 - Z_1 \sum_{j=1}^{N} q_2^{(j,k)} - Z_2 \sum_{j=1}^{N} q_1^{(i,j)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)} \right)}_{(1)} \quad (3)$$

(1) Accumulate products in `int32 += uint8 * uint8`

- quantize bias-vectors by not uint8 but int32
  - reason: quantization errors in bias-vectors tend to be overall errors because their elements are added to many output activations

(2) Scale down int32_t to uint8

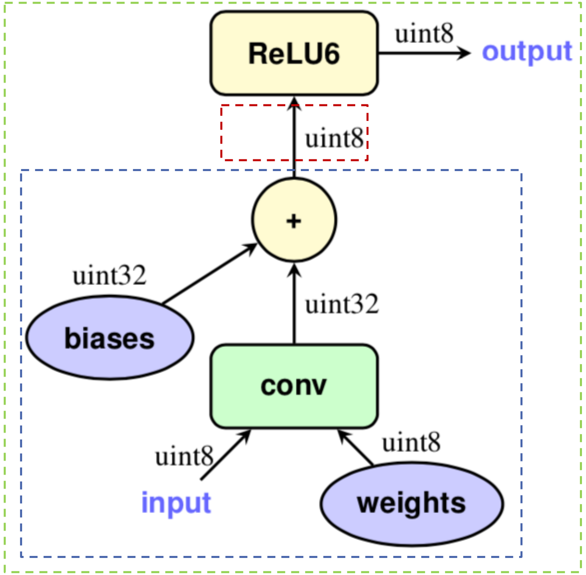
a. multiplying the fixed-point value M_0

b. n bit-shift

c. saturating cast to [0, 255]

(3) Apply activation functions

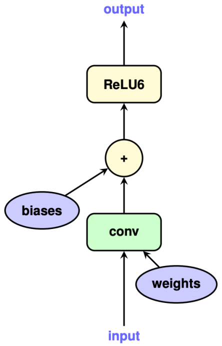
- mere clamp uint8_t because MobileNets use only ReLU and ReLU6



Quantization-aware training

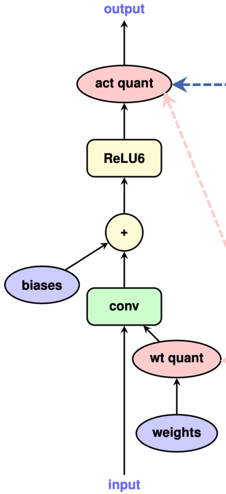
- Motivation: post-quantization has difficulties in handling:
  - large differences (> 100×) in ranges of weights for each output channels
  - outlier weight values
- Approach: simulate integer-quantization effects during training

Step-1: Create a floating-point graph as usual



After training proceeds…

Step-2: Insert *fake quantization* operations, which downcast tensors to fewer bits in float



As for activations, in addition to simulation, aggregate them via exponential moving averages (EMA)

$$\mathrm{clamp}(r;a,b) := \min\left(\max(x,a),b\right)$$

$$s(a,b,n) := \frac{b-a}{n-1}$$

$$q(r;a,b,n) := \left\lfloor \frac{\mathrm{clamp}(r;a,b)-a}{s(a,b,n)} \right\rceil s(a,b,n)+a$$

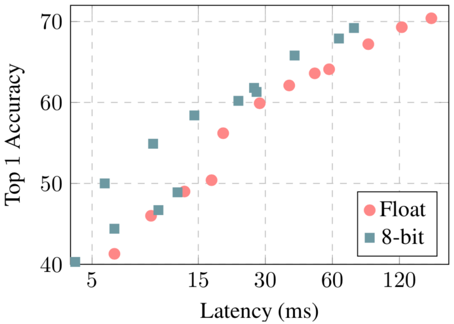
Experiments with MobileNets

- CPU: Snapdragon 835
  - march: ARM big.LITTLE [Cortex-(A73|A53)]
  - optimize by ARM NEON

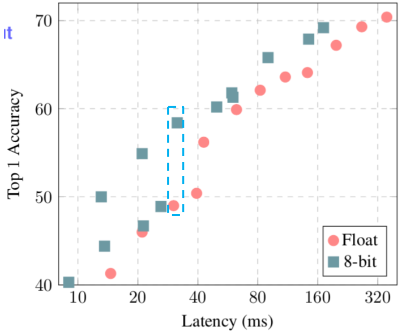
Google
Pixel 2

ImageNet:

- in the LITTLE core, the accuracy gap at 33ms (30FPS) is quite substantial (~10%)



big



LITTLE

COCO:

- MobileNet SSD was used
- up to a 50% reduction in inference time with a minimal loss in accuracy (−1.8% relative)
- The INT8 quantization deals well with regression tasks

DM	Type	mAP	LITTLE (ms)	big (ms)
100%	floats	22.1	778	370
	8 bits	21.7	687	272
50%	floats	16.7	270	121
	8 bits	16.6	146	61

Summary & My perspective

- Summary
  - integer quantization benefits in common hardware like CPU
  - quantization-aware training is crucial in quantizing modern light-weight architectures (e.g. Mobile-Nets) and error-sensitive tasks such as regression
- My perspective
  - integer quantization is the most moderate and generic scheme so far
  - Extreme quantization like BNNs is achievable only if parallel development of hardware and software works
    - Google
    - NVIDIA
    - Apple

DESIGNLINES | AI & BIG DATA DESIGNLINE

Apple Reportedly Acquires Xnor

By Sally Ward-Foxton 01.16.2020 0

Share Post Share on Facebook Share on Twitter in

Seattle startup runs AI on tiny embedded devices

Reports are circulating that the Seattle-based AI at the edge company Xnor has been quietly a
Apple. An investigation by GeekWire suggests the deal was worth in the region of $200 millio
development could mean Xnor's low-power algorithms for object detection in photos end up

Xnor, a spin-out from the Allen Institute for Artificial Intelligence (AI2), had raised $14.6 mill
since it was founded three years ago. Xnor's founders, Ali Farhadi and Mohammed Rastegari,
of YOLO, a well-known neural network widely used for object detection.