Quantisation

Efficient implementation of convolutional neural networks

Philip Leong

Computer Engineering Lab
The University of Sydney

July 2018 / PAPAA Workshop

Australia

Population: 24M (2016)
Europe: 741M (2016)
Hong Kong: 7M (2016)
Area 1/25th Tasmania

Outline

Outline

Introduction

- There are several degrees of freedom to explore when optimising DNNs
  - NN architecture (SqueezeNet, MobileNet)
  - Compression (SVD, Deep Compression, Circulant)
  - Quantization (FP16, TF-Lite, FINN, DoReFa-Net)
- This talk: quantisation

Unsigned Numbers

$$
\begin{aligned}
U &= (u_{W-1}u_{W-2}\ldots u_0), u_i \in \{0,1\} \\
&= \sum_{i=0}^{W-1} u_i 2^i
\end{aligned}
$$

- U is a W-bit unsigned integer
- Range $[0, 2^W)$

Two's Complement Numbers

$$
\begin{aligned}
X &= (x_{W-1}x_{W-2}\ldots x_0), x_i \in \{0,1\} \\
&= -x_{W-1}2^{W-1} + \sum_{i=0}^{W-2} x_i 2^i
\end{aligned}
$$

- X is a W-bit signed integer
- Range $[-2^{W-1}, 2^{W-1})$

Two's Complement Fractions

$$
\begin{aligned}
Y &= (\overbrace{y_{W-1}\ldots y_F}^{\text{I-bit integer}}\overbrace{y_{F-1}\ldots x_0}^{\text{F-bit fraction}}), y_i \in \{0,1\} \\
&= 2^{-F} \times (-x_{W-1}2^{W-1} + \sum_{i=0}^{W-2} x_i 2^i)
\end{aligned}
$$

- Y is a W-bit signed fraction with F-bit fraction
- Are two's complement numbers scaled by 2^{-F}
- Notation used: (I,F) (with $I + F = W$)
  - (W,0) same as two's complement integers
  - (1,W-1) has range [-1,1) and multiplication never overflows

Dynamic Fixed Point [CBD14]

$$D = (-1)^S . 2^{-F} \sum_{i=0}^{W-2} x_i 2^i$$

- D is dynamic fixed point number with sign bit S, fractional length F, W is word length
- Sign-magnitude fraction with F being shared within a group
- Allows number format to be adapted to different network segments e.g. layer inputs, weights and outputs can have different F

Operations on Two's Complement Fractions

- Addition and subtraction same as two's complement
- Multiplication
  - An (I,F) multiplication gives a (2I,2F) result, need to discard F bits
  - For (1,3)

$$
\begin{aligned}
0.75 \times 0.75 &= 0.110 \times 0.110 \\
&= 00.100100 \qquad \text{in (2I,2F) format} \\
&\approx 0.100 \qquad \text{in (I,F) format (truncated)}
\end{aligned}
$$

  - Integer part controls range
  - Fractional part controls spacing between numbers

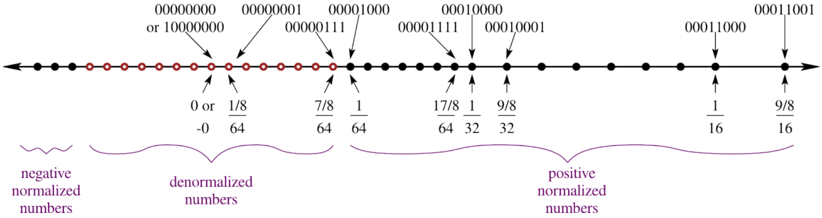
Floating Point 1

$$Z = (\overbrace{a_0}^{A}\overbrace{b_{J-1}\dots b_0}^{B}\overbrace{c_{F-1}\dots c_0}^{C}), (a_i, b_i c_i) \in \{0, 1\}$$

- Treating A, B and C as unsigned integers
  - The sign bit is $S = \begin{cases} +1 & \text{if } a_0 = 0 \\ -1, & \text{otherwise} \end{cases}$
  - The exponent is stored in a biased representation with $E = B - (2^{J-1} - 1)$
  - For normalised numbers, $B \neq 0$, and M is a positive (1,F) two's complement fraction $M = 1 + C2^{-F}$
  - For denormalised numbers $B = 0$ and there is no implicit 1 in the positive (0,F) two's complement fraction $M = C2^{-F}$

Floating Point 2

$$
Z = \begin{cases} S \times 2^E \times M & \text{if } (0 < B < 2^J - 1) \\ S \times 2^E \times (M - 1) & \text{if } (B = 0) \\ S \times \infty & \text{if } (B = 2^J - 1 \text{ and } C = 0) \\ \text{NaN} & \text{if } (B = 2^J - 1 \text{ and } C \neq 0) \end{cases}
$$



Operations on Floating Point Numbers

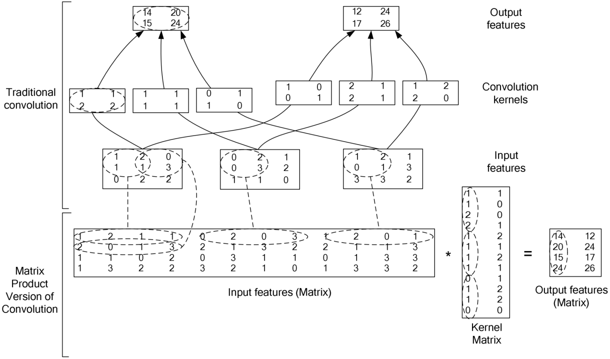
- Much larger resource utilisation
- Longer latency
- We will focus on fixed point

Outline

1. Introduction
   - Number Systems
   - Convolutional Neural Networks
   - Integer Quantisation
   - SYQ: Low Precision DNN Training
   - FINN: A Binarised Neural Network
2. Tutorial

Convolution Layer as MM

- Convolution layers converted to GEMM [CPS06]
- Efficient BLAS libraries can be exploited



DNN Computation

Computational problem in DNNs is to compute a number of dot products

$$h = g(\boldsymbol{w}^T \boldsymbol{x}) \tag{1}$$

where

- g is an element-wise nonlinear activation function
- $\boldsymbol{x} \in \mathbb{R}^{i.w.h}$ is the input vector
- $\boldsymbol{w} \in \mathbb{R}^{i.w.h}$ is the weight vector

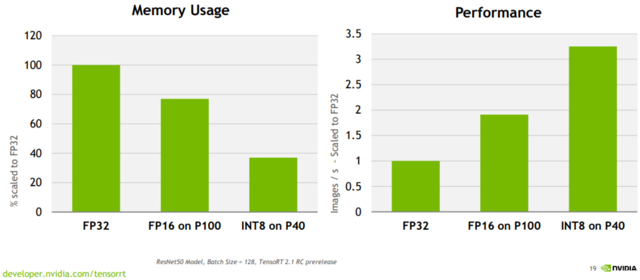
Arithmetic Intensity

- Computation of a DNN layer is MV multiplication
- For MV multiply is $O(1)$, for MM is $O(b)$ where b is block size
- Efficient CPU/GPU implementations use batch size $\gg 1$ (process a number of inputs together)
- For latency-critical applications (e.g. object detection for self-driving car), we want a batch size of 1
- Make sure comparisons are at the same batch size!

Outline

Role of Wordlength on Performance

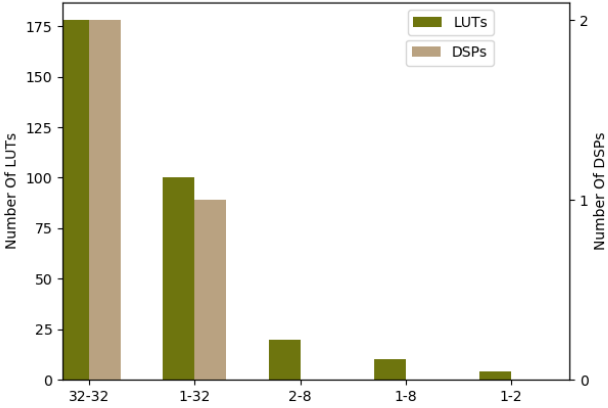
- CPU/GPU
  - Floating point performance comparable to fixed
  - Integer data types usually vectorisable hence faster
  - Nvidia offers FP64, FP32 and FP16 ($>$ Tegra X1 and Pascal)
- FPGA
  - Datapath is flexible
  - No floating point unit so fixed point normally preferred



THE UNIVERSITY OF SYDNEY

Role of Wordlength on Resources

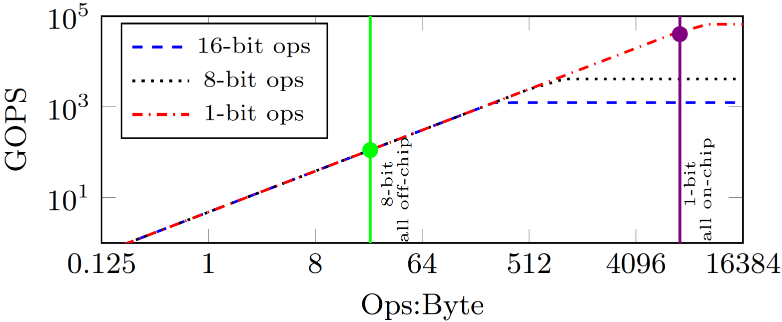
- X axis is bitwidth (weight-activation) and Y axis Number of LUTs/DSPs for MAC
- For k-bits, area is $\mathcal{O}(k^2)$



Roofline Model

Roofline model for Xilinx ZU19EG

- X axis is computational intensity (ops to perform / byte fetch), Y axis is performance
- Diagonal parts show memory-bandwith limited space
- Horizontal parts show computation limited space
- Actually this is a better metric to optimise than say GOPs/s
- Low precision extremely advantageous for performance



Integer Quantization [Jac+18]

A way to map numbers $r \in \mathbb{R}$ to unsigned integers $q \in \mathbb{U}+$ is via an affine transformation

$$r = S(q - Z) \tag{2}$$

- $\mathbb{U}+$ is the set of unsigned W-bit integers
- S, Z are the quantisation parameters
  - $S \in \mathbb{R}+$ represents a scaling constant
  - $Z \in \mathbb{U}+$ represents a zero-point

Integer MM [Jac+18]

- $N \times N$ MM defined as

$$r_3^{(i,k)} = \sum_{j=1}^{N} r_1^{(i,j)} r_2^{(j,k)}, \tag{3}$$

substituting $r = S(q - Z)$ (2) and rewriting we get

$$q_3^{(i,k)} = Z_3 + M\left(NZ_1Z_2 - Z_1 a_2^{(k)} - Z_2 \overline{a}_1^{(i)} + \sum_{j=1}^{N} q_1^{(i,j)} q_2^{(j,k)}\right) \tag{4}$$

- Multiplication with $M = \frac{S_1 S_2}{S_3}$ is implemented in (high-precision) two's complement fixed point
- $a_2^{(k)}$ and $\overline{a}_1^{(i)}$ together only take $2N^2$ additions
- Sum in (4) takes $2N^3$ and is a standard integer MAC
- CPU implementation uses uint8, accumulated as int32

THE UNIVERSITY OF SYDNEY

Quantisation Range [Jac+18]

For each layer, quantisation parameterised by (a,b,n):

$$
\begin{aligned}
clamp(r; a, b) &= min(max(x, a), b) \\
s(a, b, n) &= \frac{b - a}{n - 1}
\end{aligned}
$$

$$
q(r; a, b, n) = rnd(\frac{clamp(r; a, b) - a}{s(a, b, n)})s(a, b, n) + a \qquad (5)
$$

where $r \in \mathbb{R}$ is number to be quantised, [a,b] is quantisation range, n is number of quantisation levels and $rnd()$ rounds to nearest integer

Figure from [Jac+18] (with permission)

Training Algorithm [Jac+18]

1. Create training graph of the floating-point model
2. Insert quantisation operations for integer computation in inference path using (5)
3. Train with quantised inference but floating-point backpropagation until convergence
4. Use weights thus obtained for low-precision inference

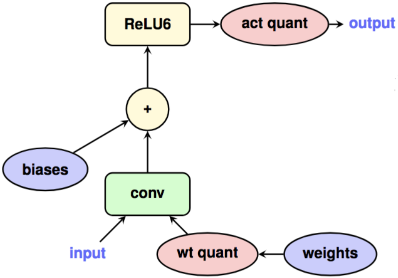


Figure from [Jac+18] (with permission)

Accuracy vs Precision [Jac+18]

ResNet50 on ImageNet, comparison with other approaches

Scheme	BWN	TWN	INQ	FGQ	Ours
Weight bits	1	2	5	2	8
Activation bits	float32	float32	float32	8	8
Accuracy	68.7%	72.5%	74.8%	70.8%	74.9%

Table 4.2: ResNet on ImageNet: Accuracy under various quantization schemes, including binary weight networks (BWN [21, 15]), ternary weight networks (TWN [21, 22]), incremental network quantization (INQ [33]) and fine-grained quantization (FGQ [26])

Figure from [Jac+18] (with permission)

Accuracy vs Latency [Jac+18]

ImageNet classifier on Google Pixel 2 (Qualcomm Snapdragon 835 big cores)

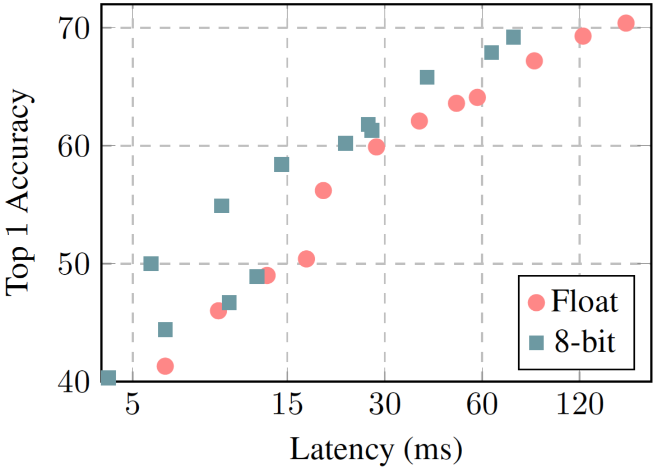


Figure from [Jac+18] (with permission)

Outline

THE UNIVERSITY OF SYDNEY

Symmetric Quantisation (SYQ) [Far+18]

- To compute quantised weights from FP weights

$$\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l \qquad (6)$$

with,

$$M_{l_{i,j}} = \begin{cases} 1 & \text{if} \quad \left|W_{l_{i,j}}\right| \geq \eta_l \\ 0 & \text{if} \quad -\eta_l < W_{l_{i,j}} < \eta_l \end{cases} \qquad (7)$$

$$sign(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ \text{-1} & \text{otherwise} \end{cases} \qquad (8)$$

where $\boldsymbol{M}$ represents a masking matrix, η is the quantization threshold hyperparameter (0 for binarised)

Symmetric Quantisation (SYQ) [Far+18]

- Make approximation $W_l \approx \alpha_l Q_l$, $Q_l \in \mathcal{C}$
- $\mathcal{C}$ is the codebook, $\mathcal{C} \in \{C_1, C_2, \ldots\}$ e.g. $\mathcal{C} = \{-1, +1\}$ for binary, $\mathcal{C} = \{-1, 0, +1\}$ for ternary
- A diagonal matrix $\boldsymbol{\alpha_l}$ is defined by the vector $\boldsymbol{\alpha_l} = \left[\alpha_l^1, ..., \alpha_l^m\right]$:

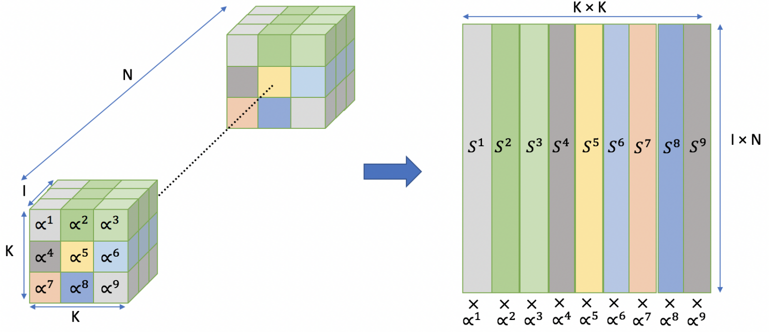
$$\boldsymbol{\alpha} = diag(\boldsymbol{\alpha}) := \begin{bmatrix} \alpha^1 & 0 & .. & 0 & 0 \\ 0 & \alpha^2 & .. & : & 0 \\ : & : & .. & \alpha^{m-1} & : \\ 0 & 0 & .. & 0 & \alpha^m \end{bmatrix}$$

- Train by solving

$$\alpha_l^* = argmin_\alpha E(\alpha, \boldsymbol{Q}) \quad s.t. \quad \alpha \geq 0,\ Q_{l_{i,j}} \in \mathcal{C}$$

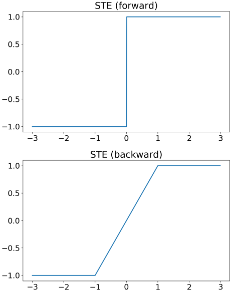
Subgroups

- Finer-grained quantisation improves weight approximation
- Pixel-wise shown, layer-wise has similar accuracy



Dealing with Non-differentiable Functions

- Recall (6) $\boldsymbol{Q}_l = sign(\boldsymbol{W}_l) \odot \boldsymbol{M}_l$
- This step function has a derivative which is zero everywhere: *vanishing gradients* problem
- Address via a straight through estimator (STE)
- Consider $q = sign(r)$ and $g_r \approx \frac{\partial C}{\partial q}$ then $\frac{\partial C}{\partial r} \approx g_q 1_{|r| \leq 1}$



Results for 8-bit activations

Model		Bin	Tern	FP32	Reference
AlexNet	Top-1	**56.6**	**58.1**	56.6	57.1
	Top-5	**79.4**	**80.8**	80.2	80.2
VGG	Top-1	**66.2**	**68.7**	69.4	-
	Top-5	**87.0**	**88.5**	89.1	-
ResNet-18	Top-1	**62.9**	**67.7**	69.1	69.6
	Top-5	**84.6**	**87.8**	89.0	89.2
ResNet-34	Top-1	**67.0**	**70.8**	71.3	73.3
	Top-5	**87.6**	**89.8**	89.1	91.3
ResNet-50	Top-1	**70.6**	**72.3**	76.0	76.0
	Top-5	**89.6**	**90.9**	93.0	93.0

- Our ResNet and AlexNet reference results are obtained from https://github.com/facebook/fb.resnet.torch and https://github.com/BVLC/caffe

Alexnet Comparison

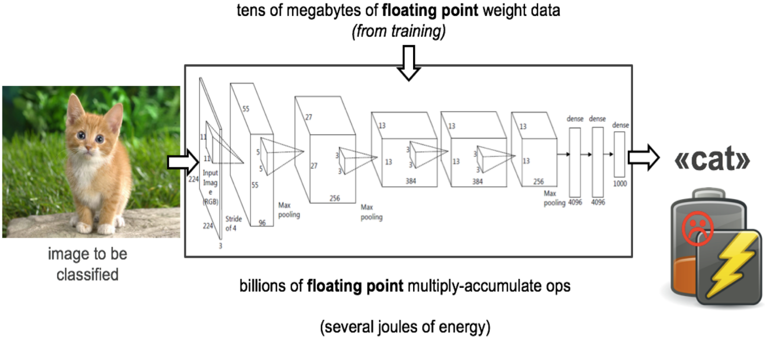
Model	Weights	Act.	Top-1	Top-5
DoReFa-Net [Zho+16]	1	2	49.8	-
QNN [Hub+16]	1	2	51.0	73.7
HWGQ [Cai+17]	1	2	52.7	76.3
SYQ	**1**	**2**	**55.2**	**78.4**
DoReFa-Net [Zho+16]	1	4	53.0	-
SYQ	**1**	**4**	**56.2**	**79.4**
BWN [Ras+16]	1	32	56.8	79.4
SYQ	**1**	**8**	**56.6**	**79.4**
SYQ	**2**	**2**	**55.7**	**79.1**
FGQ [Mel+17]	2	8	49.04	-
TTQ [Zhu+16]	2	32	57.5	79.7
SYQ	**2**	**8**	**58.1**	**80.8**

THE UNIVERSITY OF SYDNEY

Outline

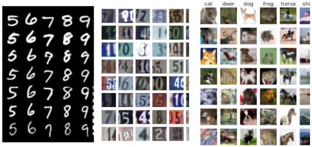
Inference with Convolutional Neural Networks

Slides from Yaman Umuroglu et. al., “FINN: A framework for fast, scalable binarized neural network inference,” FPGA’17



Binarized Neural Networks

- The extreme case of quantization
  - Permit only two values: +1 and -1
  - Binary weights, binary activations
  - Trained from scratch, not truncated FP



- Courbariaux and Hubara et al. (NIPS 2016)
  - Competitive results on three smaller benchmarks
  - Open source training flow
  - Standard “deep learning” layers
    - Convolutions, max pooling, batch norm, fully connected…

	MNIST	SVHN	CIFAR-10
Binary weights & activations	0.96%	2.53%	10.15%
FP weights & activations	0.94%	1.69%	7.62%
BNN accuracy loss	-0.2%	-0.84%	-2.53%

% classification error (lower is better)

Advantages of BNNs

Vivado HLS estimates on Xilinx UltraScale+ MPSoC ZU19EG

› *Much* smaller datapaths
  - Multiply becomes XNOR, addition becomes popcount
  - No DSPs needed, everything in LUTs
  - Lower cost per op = more ops every cycle

› *Much* smaller weights
  - Large networks can fit entirely into on-chip memory (OCM)
  - More bandwidth, less energy compared to off-chip

Precision	Peak TOPS	On-chip weights
1b	~66	~70 M
8b	~4	~10 M
16b	~1	~5 M
32b	~0.3	~2 M

200x

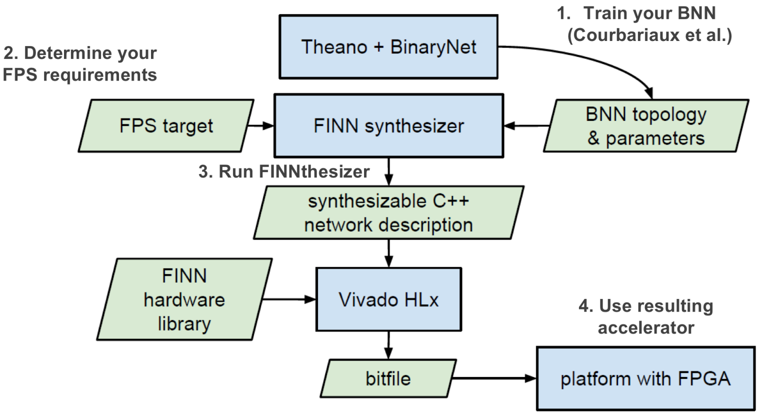
30x

› **fast** inference with **large BNNs**

Design Flow

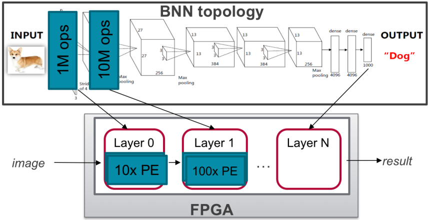
- One size does not fit all - Generate tailored hardware for network and use-case
- Stay on-chip - Higher energy efficiency and bandwidth
- Support portability and rapid exploration - Vivado HLS (High-Level Synthesis)
- Simplify with BNN-specific optimizations - Exploit compile time optimizations to simplify hardware, e.g. batchnorm and activation => thresholding

Design Flow



1. Train your BNN (Courbariaux et al.)
2. Determine your FPS requirements
Theano + BinaryNet
FPS target
FINN synthesizer
BNN topology & parameters
3. Run FINNthesizer
synthesizable C++ network description
FINN hardware library
Vivado HLx
4. Use resulting accelerator
bitfile
platform with FPGA

Heterogeneous Streaming Architecture



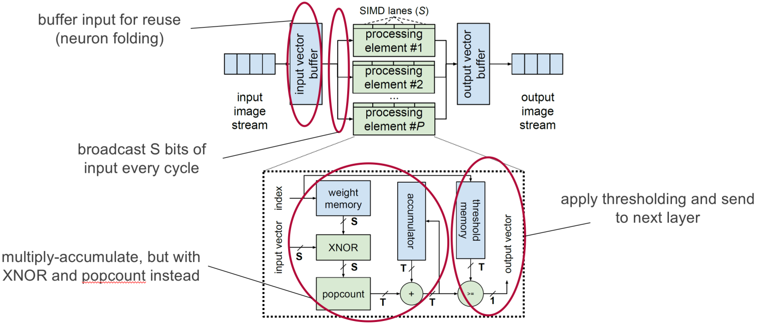
1x FPS

10x FPS

- **One hardware layer per BNN layer, parameters built into bitstream**
  - Both inter- and intra-layer parallelism
- **Heterogeneous: Avoid "one-size-fits-all" penalties**
  - Allocate compute resources according to FPS and network requirements
- **Streaming: Maximize throughput, minimize latency**
  - Overlapping computation and communication, batch size = 1

Matrix-Vector Threshold Unit (MVTU)

- **Core computational element of FINN, tiled matrix-vector multiply**
- **Computes a (P) row x (S) column chunk of matrix every cycle, per-layer configurable tile size**



Slide © Copyright 2016 Xilinx

Convolutional Layers

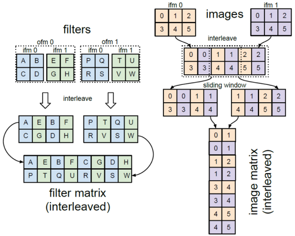
- **Lower convolutions to matrix-matrix multiplication, $W \cdot I$**
  - W : filter matrix (generated offline)
  - I: image matrix (generated on-the-fly)

- **Two components:**

convolution layer

slide window over image (generate I matrix)

matrix-matrix multiply, one vector at a time



Performance

		Accuracy	FPS	Power (chip)	Power (wall)	kFPS / Watt (chip)	kFPS / Watt (wall)	Precision
FINN	MNIST, SFC-max	95.8%	12.3 M	7.3 W	21.2 W	1693	583	1
	MNIST, LFC-max	98.4%	1.5 M	8.8 W	22.6 W	177	269	1
	CIFAR-10, CNV-max	80.1%	21.9 k	3.6 W	11.7 W	6	2	1
	SVHN, CNV-max	94.9%	21.9 k	3.6 W	11.7 W	6	2	1
Prior Work	MNIST, Alemdar et al.	97.8%	255.1 k	0.3 W	-	806	-	2
	CIFAR-10, TrueNorth	83.4%	1.2 k	0.2 W	-	6	-	1
	SVHN, TrueNorth	96.7%	2.5 k	0.3 W	-	10	-	1

Max accuracy loss: ~3%

10 – 100x better performance

CIFAR-10/SVHN energy efficiency comparable to TrueNorth ASIC

Summary

- Reducing precision
  - Significantly reduce computational costs in DNNs
  - Data may now fit entirely on chip, avoiding external memory accesses
  - Computations greatly simplified
  - Key dimension for optimisation in CPU/GPU/FPGA implementations
- Convolutional layer can be computed as a MM
- Still an active research topic

THE UNIVERSITY OF SYDNEY

Tutorial Question 1

1. Download VM (`quantisation_usyd.ova`) from `https://bluemountain.eee.hku.hk/papaa2018/`
2. Import to Virtualbox, and inside VM do

   ```
   git clone https://gitlab.com/phwl/syq-cifar10.git
   ```

3. Derive Equation (4) from (3) and (2)

Tutorial Question 2

1. Cifar10 is a *very small* neural network benchmark[1]. Test precision with SYQ using:

   ```
   cd syq-cifar10/src
   python cifar10_eval.py
   ```

   (this can run during training)

2. The code provided performs binary quantisation. Modify the code to determine precision for binary, ternary and floating-point (use the checkpoint files provided to initialise your training).

   ```
   python cifar10_train.py
   ```

1 https://www.tensorflow.org/tutorials/images/deep_cnn

References I

[CBD14] Matthieu Courbariaux et al. "Low precision arithmetic for deep learning". In: *CoRR* abs/1412.7024 (2014). arXiv: `1412.7024`. URL: `http://arxiv.org/abs/1412.7024`.

[CPS06] Kumar Chellapilla et al. "High Performance Convolutional Neural Networks for Document Processing". In: *Tenth International Workshop on Frontiers in Handwriting Recognition*. Ed. by Guy Lorette. http://www.suvisoft.com. Université de Rennes 1. La Baule (France): Suvisoft, Oct. 2006. URL: `https://hal.inria.fr/inria-00112631`.

References II

[Jac+18] Benoit Jacob et al. “Quantization and Training of Neural Networks for Efficient Integer-Arithmetic-Only Inference”. In: *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. June 2018.

[Far+18] Julian Faraone et al. “SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks”. In: *Proc. Computer Vision and Pattern Recognition (CVPR)*. Utah, US, June 2018.

[Zho+16] Shuchang Zhou et al. “DoReFa-Net: Training Low Bitwidth Convolutional Neural Networks with Low Bitwidth Gradients”. In: *CoRR* abs/1606.06160 (2016). URL: http://arxiv.org/abs/1606.06160.

References III

[Hub+16] Itay Hubara et al. "Quantized Neural Networks: Training Neural Networks with Low Precision Weights and Activations". In: *CoRR* abs/1609.07061 (2016). arXiv: 1609.07061. URL: http://arxiv.org/abs/1609.07061.

[Cai+17] Zhaowei Cai et al. "Deep Learning with Low Precision by Half-wave Gaussian Quantization". In: *CoRR* abs/1702.00953 (2017). arXiv: 1702.00953. URL: http://arxiv.org/abs/1702.00953.

[Ras+16] Mohammad Rastegari et al. "XNOR-Net: ImageNet Classification Using Binary Convolutional Neural Networks". In: *CoRR* abs/1603.05279 (2016). URL: http://arxiv.org/abs/1603.05279.

References IV

[Mel+17] Naveen Mellempudi et al. "Ternary Neural Networks with Fine-Grained Quantization". In: *CoRR* abs/1705.01462 (2017). URL: http://arxiv.org/abs/1705.01462.

[Zhu+16] Chenzhuo Zhu et al. "Trained Ternary Quantization". In: *CoRR* abs/1612.01064 (2016). URL: http://arxiv.org/abs/1612.01064.