Fast, Scalable Quantized Neural Network Inference on FPGAs

Lecturer: Ke-Han Li

Date: 2022/4/13

[FPGA’17: FINN: A Framework for Fast, Scalable Binarized Neural Network Inference]
(https://arxiv.org/abs/1612.07119)

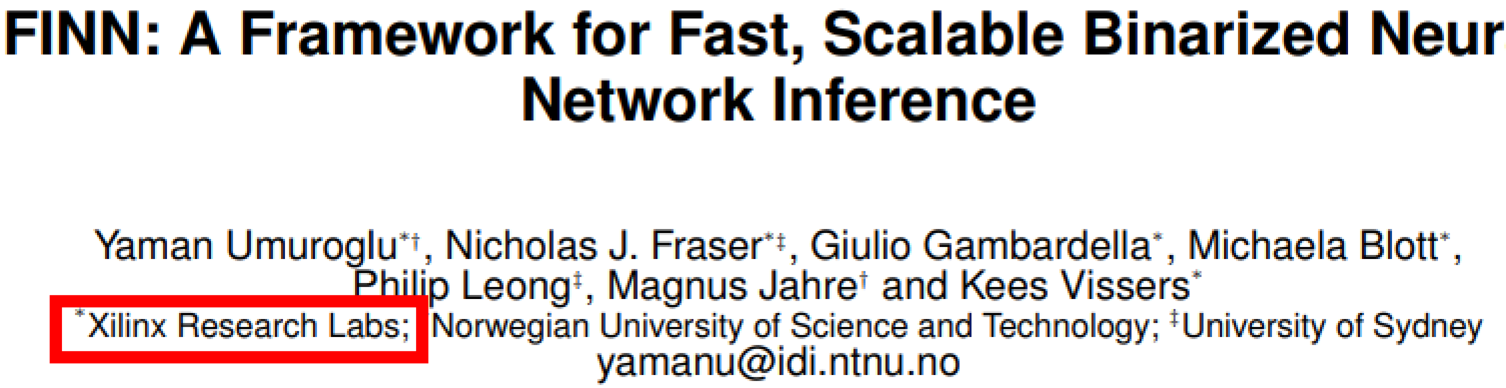
Outline

- Introduction to FINN
- Network Define
- NN Hardware
- Lab Description

Outline

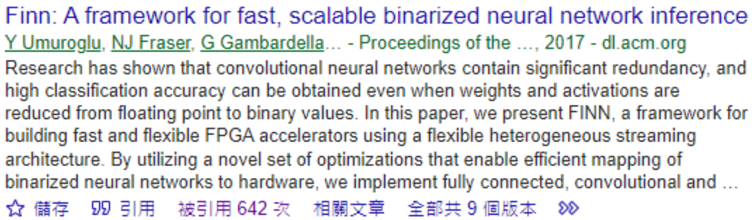
- Introduction to FINN
- Network Define
- NN Hardware
- Lab Description

FINN: The Beginning (FPGA'17)

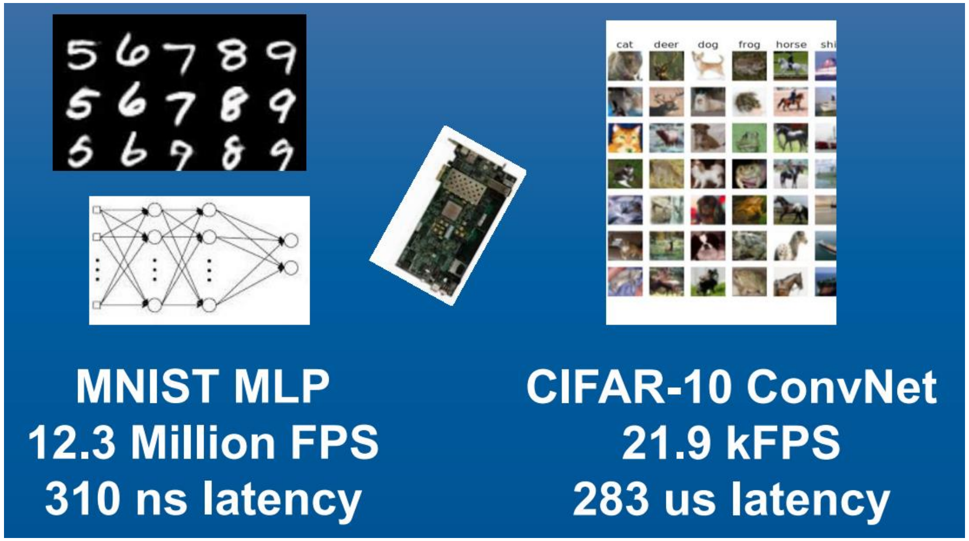


FINN: A Framework for Fast, Scalable Binarized Neural Network Inference

Yaman Umuroglu[*†], Nicholas J. Fraser[*‡], Giulio Gambardella[*], Michaela Blott[*],
Philip Leong[‡], Magnus Jahre[†] and Kees Vissers[*]
[*]Xilinx Research Labs; [†]Norwegian University of Science and Technology; [‡]University of Sydney
yamanu@idi.ntnu.no



Finn: A framework for fast, scalable binarized neural network inference
Y Umuroglu, NJ Fraser, G Gambardella… - Proceedings of the …, 2017 - dl.acm.org
Research has shown that convolutional neural networks contain significant redundancy, and high classification accuracy can be obtained even when weights and activations are reduced from floating point to binary values. In this paper, we present FINN, a framework for building fast and flexible FPGA accelerators using a flexible heterogeneous streaming architecture. By utilizing a novel set of optimizations that enable efficient mapping of binarized neural networks to hardware, we implement fully connected, convolutional and …
☆ 儲存 99 引用 被引用 642 次 相關文章 全部共 9 個版本 »

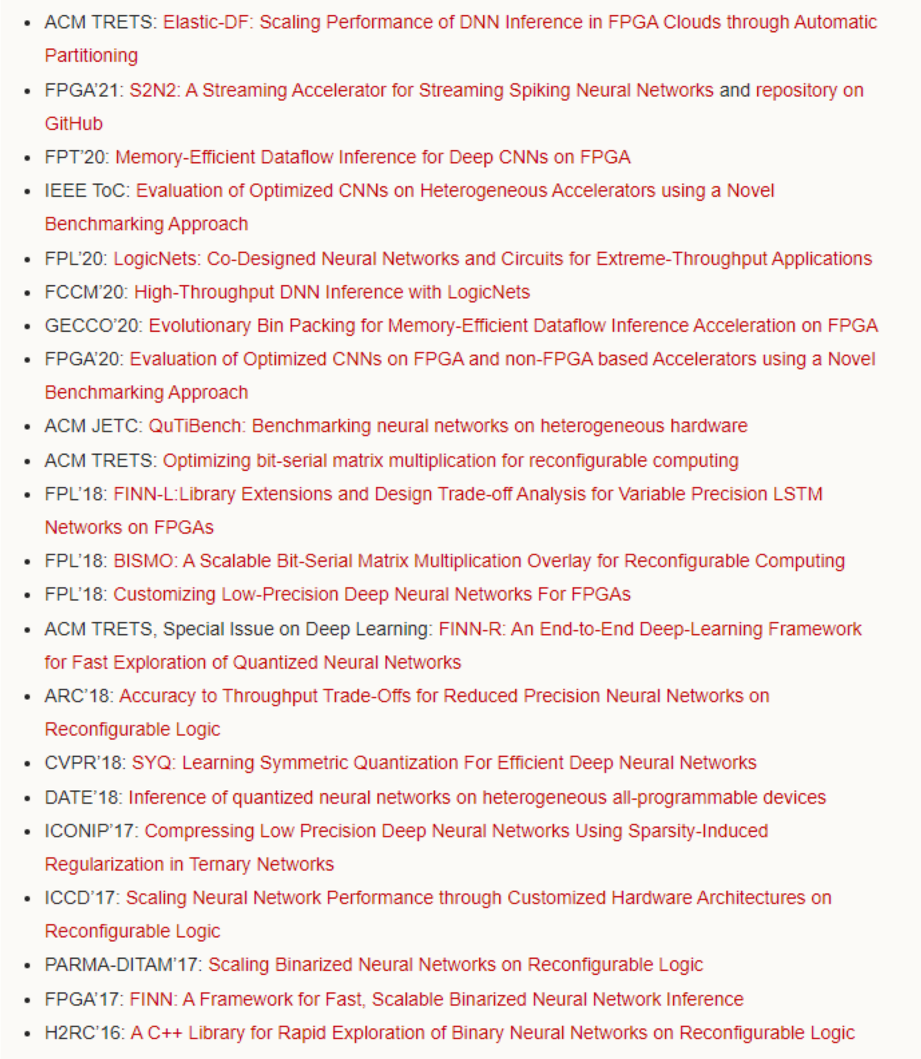


MNIST MLP
12.3 Million FPS
310 ns latency

CIFAR-10 ConvNet
21.9 kFPS
283 us latency

Publications

https://xilinx.github.io/finn/publications



- ACM TRETS: Elastic-DF: Scaling Performance of DNN Inference in FPGA Clouds through Automatic Partitioning
- FPGA'21: S2N2: A Streaming Accelerator for Streaming Spiking Neural Networks and repository on GitHub
- FPT'20: Memory-Efficient Dataflow Inference for Deep CNNs on FPGA
- IEEE ToC: Evaluation of Optimized CNNs on Heterogeneous Accelerators using a Novel Benchmarking Approach
- FPL'20: LogicNets: Co-Designed Neural Networks and Circuits for Extreme-Throughput Applications
- FCCM'20: High-Throughput DNN Inference with LogicNets
- GECCO'20: Evolutionary Bin Packing for Memory-Efficient Dataflow Inference Acceleration on FPGA
- FPGA'20: Evaluation of Optimized CNNs on FPGA and non-FPGA based Accelerators using a Novel Benchmarking Approach
- ACM JETC: QuTiBench: Benchmarking neural networks on heterogeneous hardware
- ACM TRETS: Optimizing bit-serial matrix multiplication for reconfigurable computing
- FPL'18: FINN-L:Library Extensions and Design Trade-off Analysis for Variable Precision LSTM Networks on FPGAs
- FPL'18: BISMO: A Scalable Bit-Serial Matrix Multiplication Overlay for Reconfigurable Computing
- FPL'18: Customizing Low-Precision Deep Neural Networks For FPGAs
- ACM TRETS, Special Issue on Deep Learning: FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks
- ARC'18: Accuracy to Throughput Trade-Offs for Reduced Precision Neural Networks on Reconfigurable Logic
- CVPR'18: SYQ: Learning Symmetric Quantization For Efficient Deep Neural Networks
- DATE'18: Inference of quantized neural networks on heterogeneous all-programmable devices
- ICONIP'17: Compressing Low Precision Deep Neural Networks Using Sparsity-Induced Regularization in Ternary Networks
- ICCD'17: Scaling Neural Network Performance through Customized Hardware Architectures on Reconfigurable Logic
- PARMA-DITAM'17: Scaling Binarized Neural Networks on Reconfigurable Logic
- FPGA'17: FINN: A Framework for Fast, Scalable Binarized Neural Network Inference
- H2RC'16: A C++ Library for Rapid Exploration of Binary Neural Networks on Reconfigurable Logic

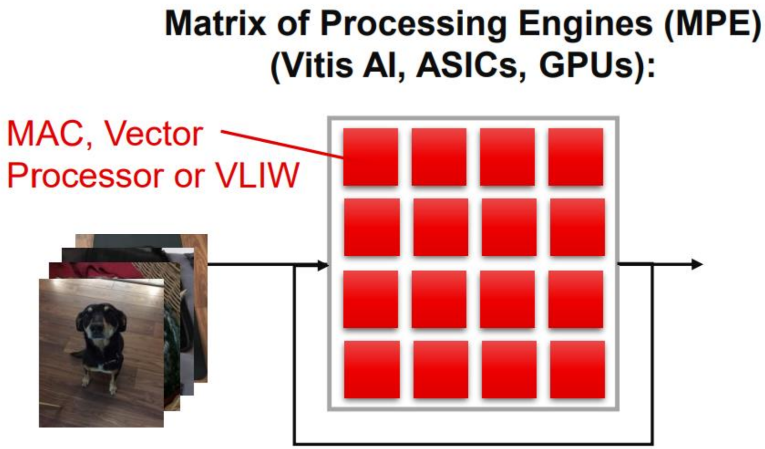
FINN – Project Mission

- Mission
  - Tools and platforms for creation of high throughput, ultra-low latency DNN compute architectures
- End-to-end flow
  - Users can easily create specialized hardware architectures on an FPGA and benefit from custom architectures and custom precision
- Open source
  - Transparency and flexibility to adapt to end-users’ applications

Two Key Techniques for Customization in FINN

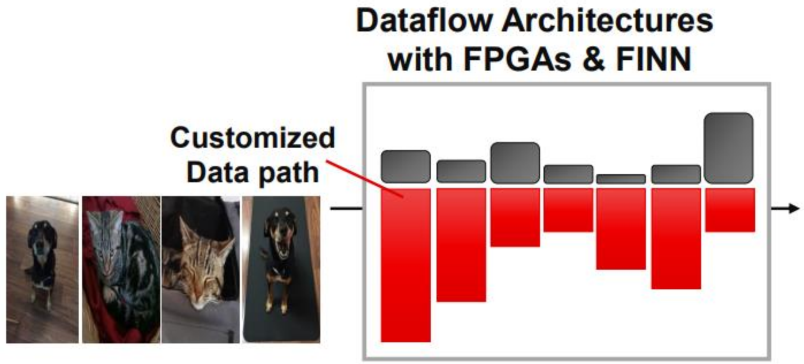
- Streaming Dataflow Architectures with FPGAs & FINN
- Custom Precision Few-bit weights & activations

Customized Dataflow Processing versus More Generic Architectures



Customized for typical DNN operations

- multiply accumulate
- Lower throughput (~10KRps)
- Flexibility for ASICs
- Applications: CV, Speech

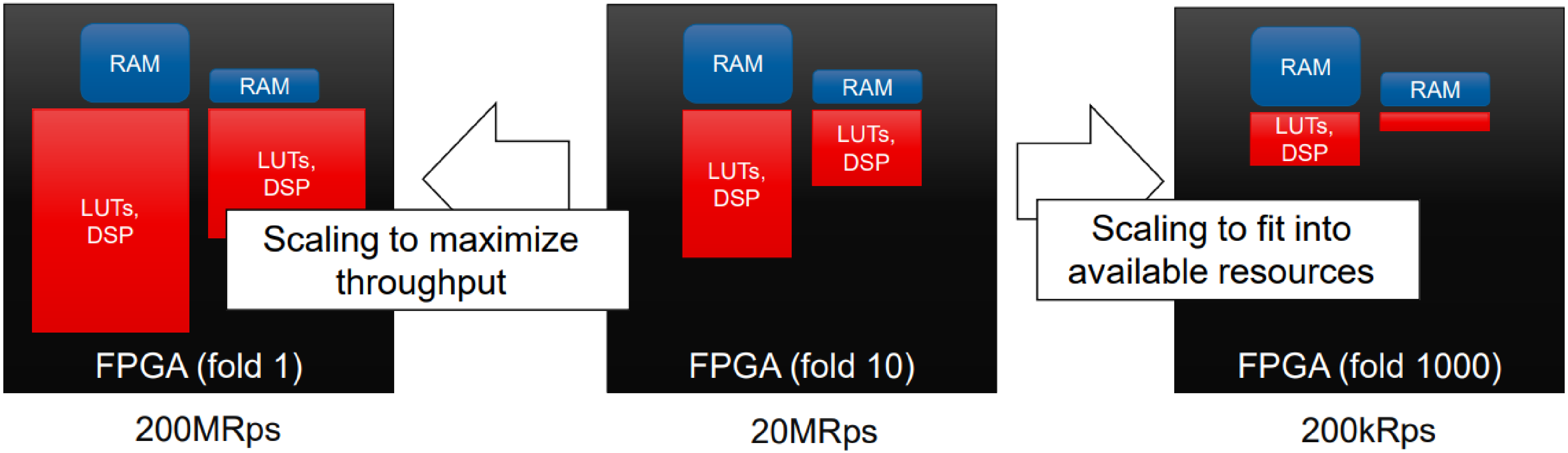


Customized/adapt for specific DNN topologies

- Streaming interfaces
- Specialization -> higher efficiency
- Lower latency (no intermediate buffering)
- Higher throughput (~100MRps)
- Flexibility through reconfiguration
- Applications: smaller DNNs

Dataflow Processing

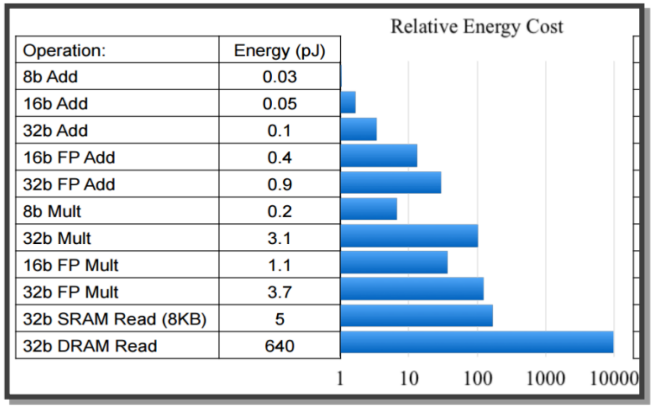
- Scale performance & resources to meet the application requirements
- If resources allow, we can completely unfold to create a circuit that inferences at clock



Customizing Arithmetic to Minimum Precision Required

- Reducing precision shrinks hardware cost/ scales performance
  - Instantiate n-times more compute within the same fabric, thereby scale performance n-times
  - 8b/8b -> 1b/1b, RTL => 70x
- Potential to reduce memory footprint
  - NN model can stay on-chip => no memory bottlenecks
- Inherently saves power

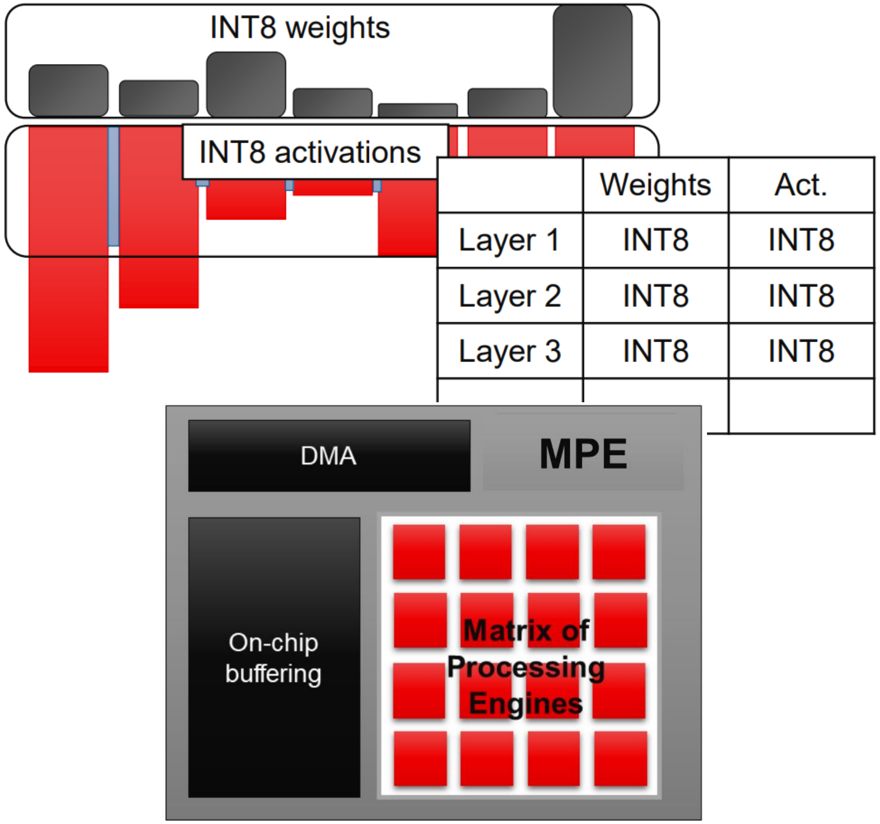
Precision	Modelsize [MB] (ResNet50)
1b	3.2
8b	25.5
32b	102.5


Relative Energy Cost

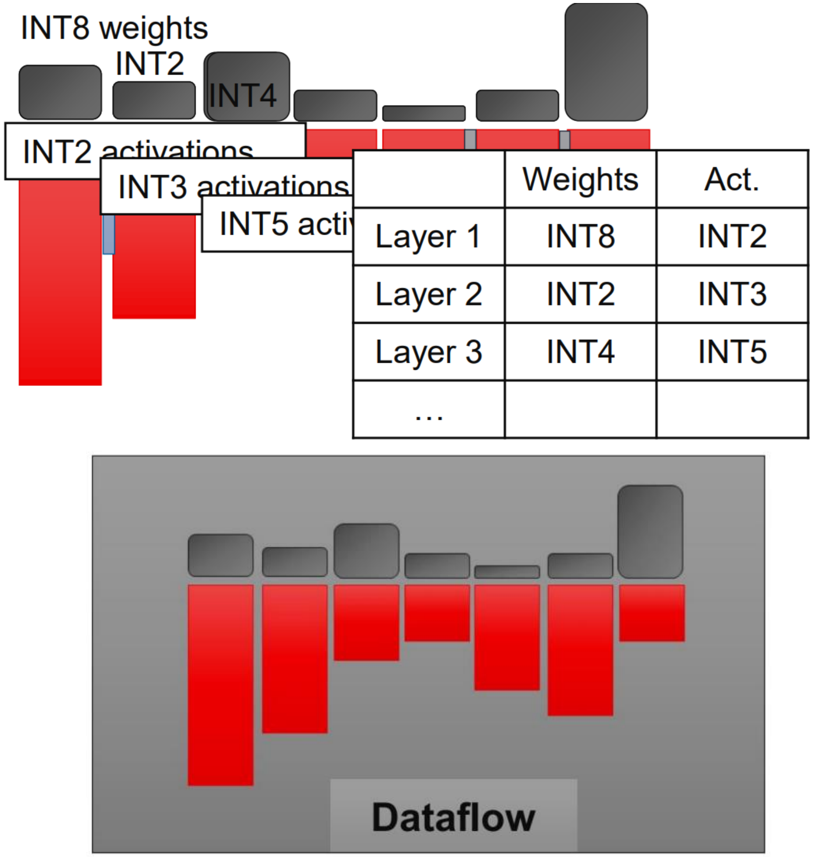
Operation:	Energy (pJ)
8b Add	0.03
16b Add	0.05
32b Add	0.1
16b FP Add	0.4
32b FP Add	0.9
8b Mult	0.2
32b Mult	3.1
16b FP Mult	1.1
32b FP Mult	3.7
32b SRAM Read (8KB)	5
32b DRAM Read	640

Granularity of Customizing Arithmetic

- Dataflow architectures can exploit custom arithmetic at a greater degree



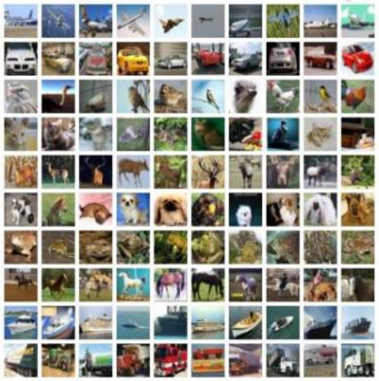
	Weights	Act.
Layer 1	INT8	INT8
Layer 2	INT8	INT8
Layer 3	INT8	INT8



	Weights	Act.
Layer 1	INT8	INT2
Layer 2	INT2	INT3
Layer 3	INT4	INT5
…		

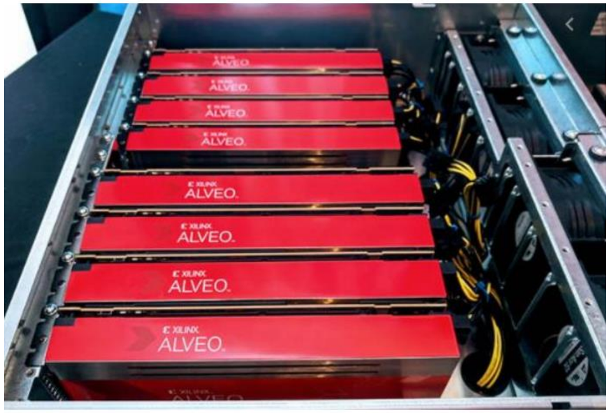
Few-bit DNNs + FPGA Dataflow: Showcases

Low-Power, Real-Time Image Classification



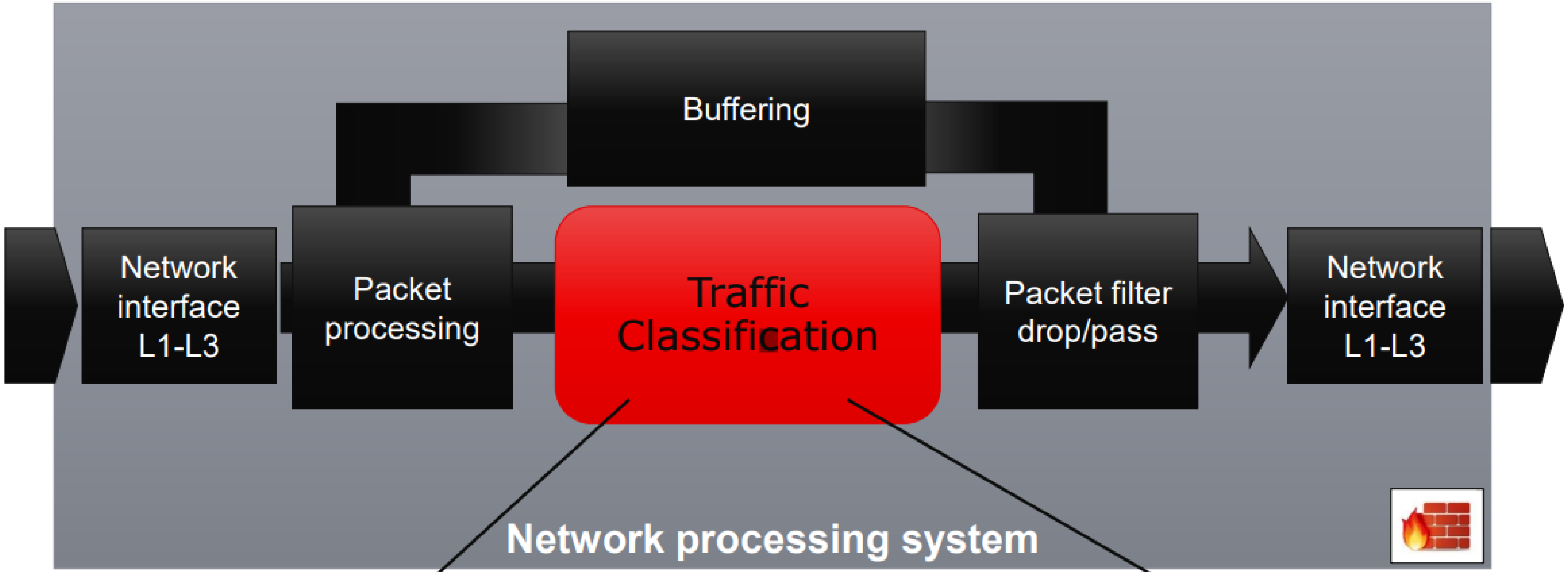
CIFAR-10 CNV on PYNQ-Z1
3kFPS @ 2.5 W
1ms latency

Single and multi-node ImageNet Classification



MNv1: 5.9kFPS, 2.2 msec (2x U280)
RN50: 3.1kFPS, 1.7msec (1x U250)

Deep Network Intrusion Detection System (NIDS)



UNSW-NB15 dataset

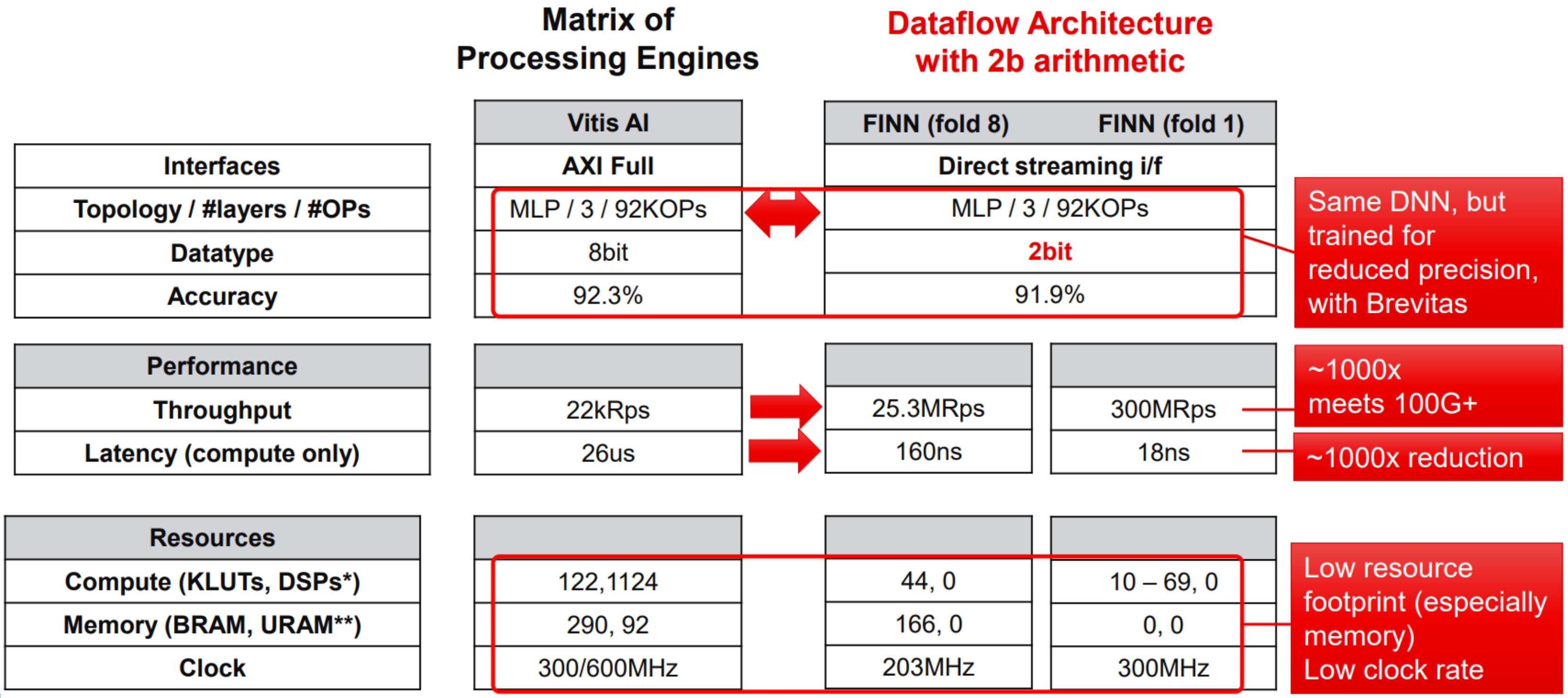
DNNs are increasingly popular:
- increased accuracy
- avoiding feature engineering

Throughput: 150MRps for 100G line rate
Latency sensitive (100Mb/msec)

MRps: Million requests per second
Assuming 64B / packet

Deep Network Intrusion Detection System (NIDS) Results

- 1000x performance improvement over Vitis AI, less resources, 100Gbps line rate (150MRps)
- Through dataflow processing, reduced precision



	Matrix of Processing Engines	Dataflow Architecture with 2b arithmetic	
	Vitis AI	FINN (fold 8)	FINN (fold 1)
Interfaces	AXI Full	Direct streaming i/f	
Topology / #layers / #OPs	MLP / 3 / 92KOPs	MLP / 3 / 92KOPs	
Datatype	8bit	2bit	
Accuracy	92.3%	91.9%	
Performance			
Throughput	22kRps	25.3MRps	300MRps
Latency (compute only)	26us	160ns	18ns
Resources			
Compute (KLUTs, DSPs*)	122,1124	44, 0	10 – 69, 0
Memory (BRAM, URAM**)	290, 92	166, 0	0, 0
Clock	300/600MHz	203MHz	300MHz

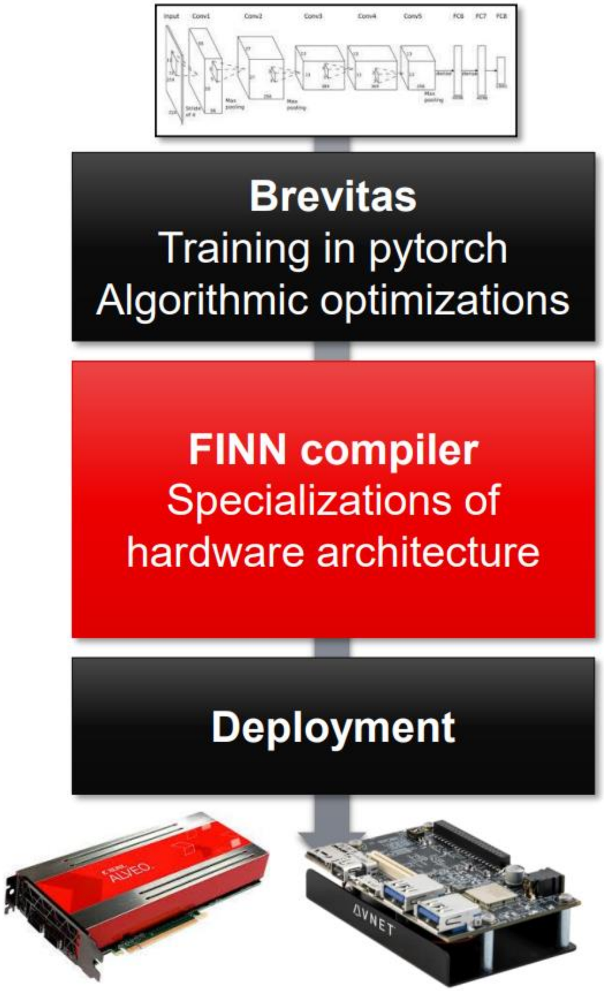
Same DNN, but trained for reduced precision, with Brevitas

~1000x meets 100G+

~1000x reduction

Low resource footprint (especially memory)
Low clock rate

FINN Framework: From DNN to FPGA Deployment



Brevitas
Training in pytorch
Algorithmic optimizations

- Train or even learn reduced precision DNNs
- Library of standard layers

FINN compiler
Specializations of
hardware architecture

- Perform optimizations
- Map to Vivado HLS
- Create DNN hardware IP

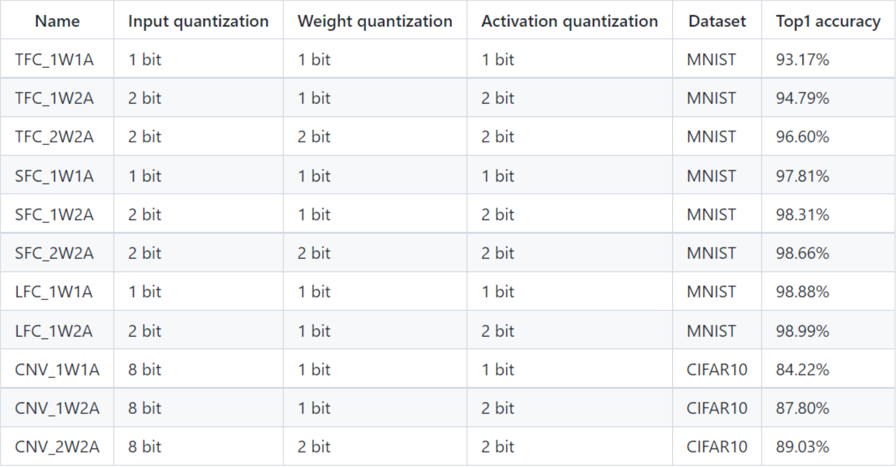
Deployment

- Works on embedded and Alveo platforms

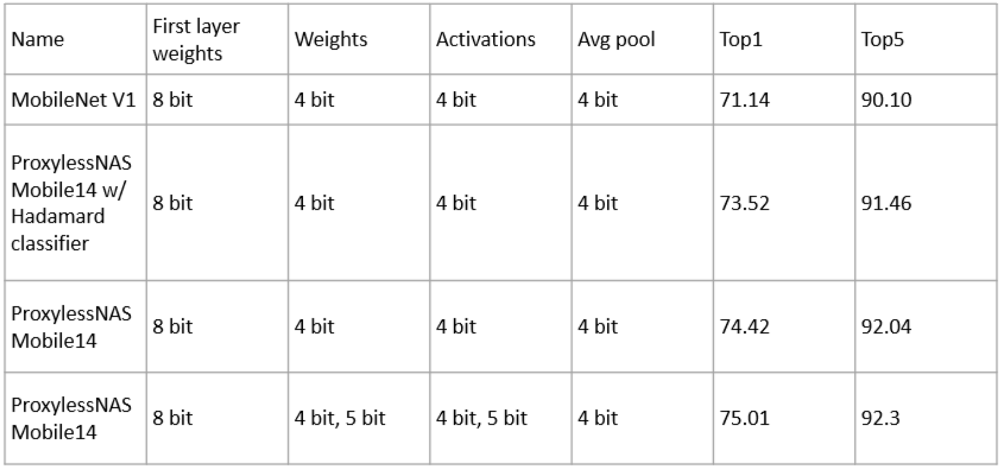
Brevitas:

A PyTorch Library for Quantization-Aware Training

- Brevitas is a PyTorch research library for quantization-aware training (QAT).
- Export to ONNX
  - To import into the FINN compiler



Name	Input quantization	Weight quantization	Activation quantization	Dataset	Top1 accuracy
TFC_1W1A	1 bit	1 bit	1 bit	MNIST	93.17%
TFC_1W2A	2 bit	1 bit	2 bit	MNIST	94.79%
TFC_2W2A	2 bit	2 bit	2 bit	MNIST	96.60%
SFC_1W1A	1 bit	1 bit	1 bit	MNIST	97.81%
SFC_1W2A	2 bit	1 bit	2 bit	MNIST	98.31%
SFC_2W2A	2 bit	2 bit	2 bit	MNIST	98.66%
LFC_1W1A	1 bit	1 bit	1 bit	MNIST	98.88%
LFC_1W2A	2 bit	1 bit	2 bit	MNIST	98.99%
CNV_1W1A	8 bit	1 bit	1 bit	CIFAR10	84.22%
CNV_1W2A	8 bit	1 bit	2 bit	CIFAR10	87.80%
CNV_2W2A	8 bit	2 bit	2 bit	CIFAR10	89.03%



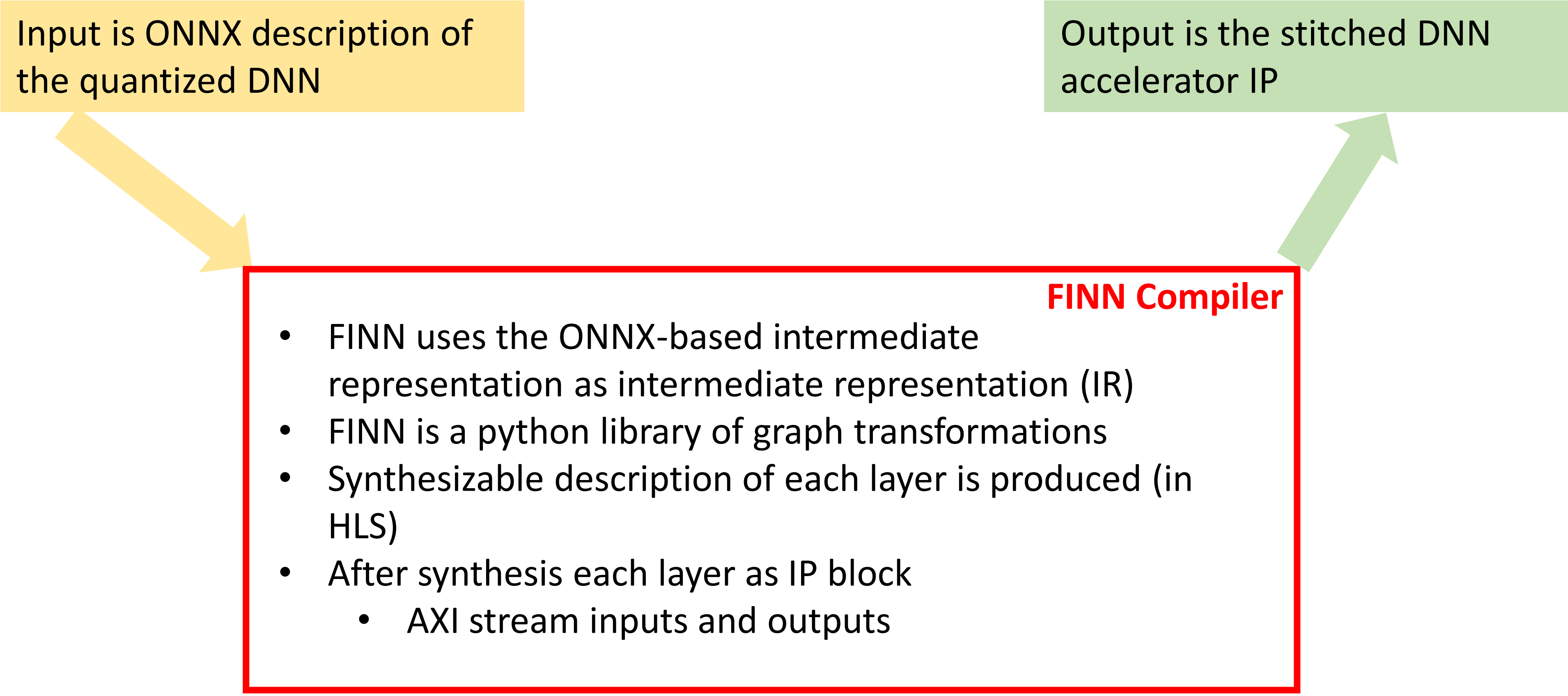
Name	First layer weights	Weights	Activations	Avg pool	Top1	Top5
MobileNet V1	8 bit	4 bit	4 bit	4 bit	71.14	90.10
ProxylessNAS Mobile14 w/ Hadamard classifier	8 bit	4 bit	4 bit	4 bit	73.52	91.46
ProxylessNAS Mobile14	8 bit	4 bit	4 bit	4 bit	74.42	92.04
ProxylessNAS Mobile14	8 bit	4 bit, 5 bit	4 bit, 5 bit	4 bit	75.01	92.3

Open Neural Network Exchange (ONNX)

- ONNX provides an open source format for AI models, both deep learning and traditional ML.
- ONNX is widely supported and can be found in many frameworks, tools, and hardware.
- Enabling interoperability between different frameworks and streamlining the path from research to production helps increase the speed of innovation in the AI community.

FINN Compiler

- Transform DNN into Custom Dataflow Architecture



Input is ONNX description of the quantized DNN

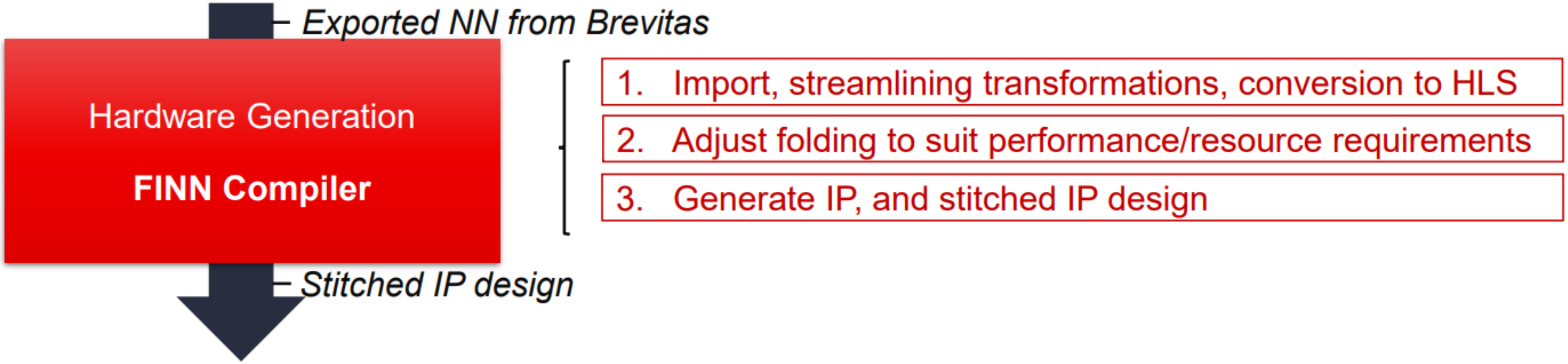
Output is the stitched DNN accelerator IP

FINN Compiler

- FINN uses the ONNX-based intermediate representation as intermediate representation (IR)
- FINN is a python library of graph transformations
- Synthesizable description of each layer is produced (in HLS)
- After synthesis each layer as IP block
  - AXI stream inputs and outputs

FINN Compiler for Hardware Generation

- In 3 Steps



Exported NN from Brevitas

Hardware Generation

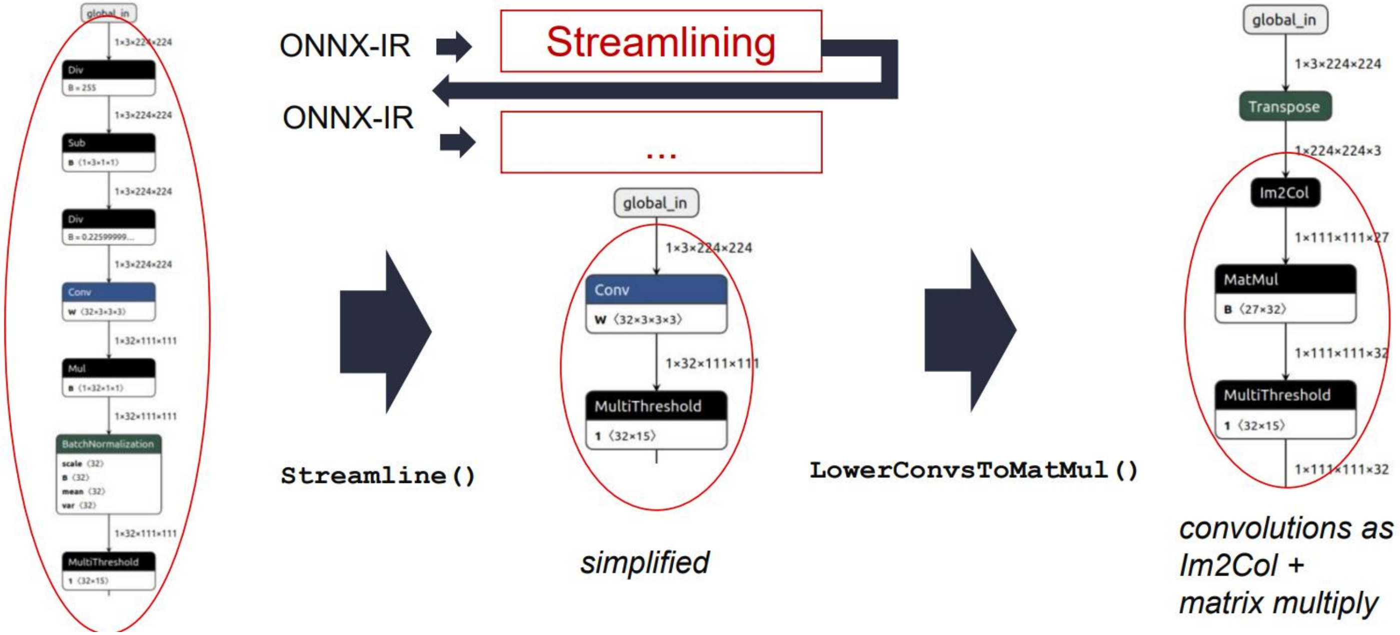
FINN Compiler

1. Import, streamlining transformations, conversion to HLS
2. Adjust folding to suit performance/resource requirements
3. Generate IP, and stitched IP design

Stitched IP design

FINN Compiler: Optimization

- Every Step is a ONNX Graph Transformations
- Optimization, lowering, code generation... are all transformations **More details next week!**



ONNX-IR → Streamlining

ONNX-IR → ...

Streamline()

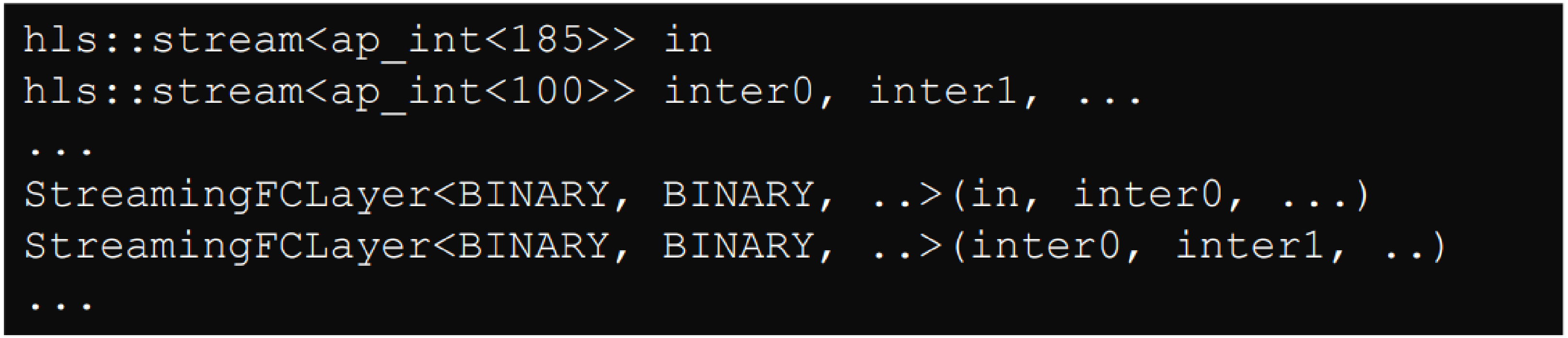
LowerConvsToMatMul()

simplified

convolutions as Im2Col + matrix multiply

FINN Compiler: HLS Generation

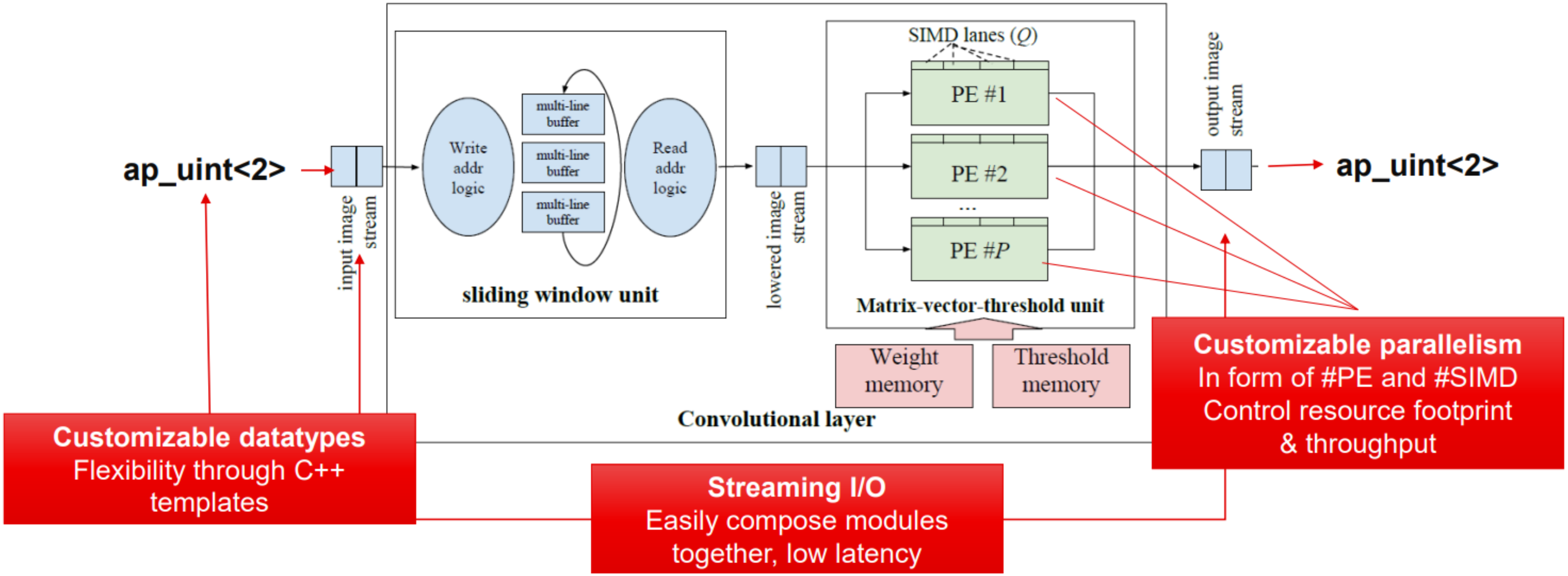
- Generate calls to a pre-optimized Vivado HLS C++ library
- Support arbitrary-precision datatypes via templates
- Synthesizable to RTL



```
hls::stream<ap_int<185>> in
hls::stream<ap_int<100>> inter0, inter1, ...
...
StreamingFCLayer<BINARY, BINARY, ..>(in, inter0, ...)
StreamingFCLayer<BINARY, BINARY, ..>(inter0, inter1, ..)
...
```

The FINN HLS Library

- Key component: MVTU (Matrix Vector Threshold Unit)



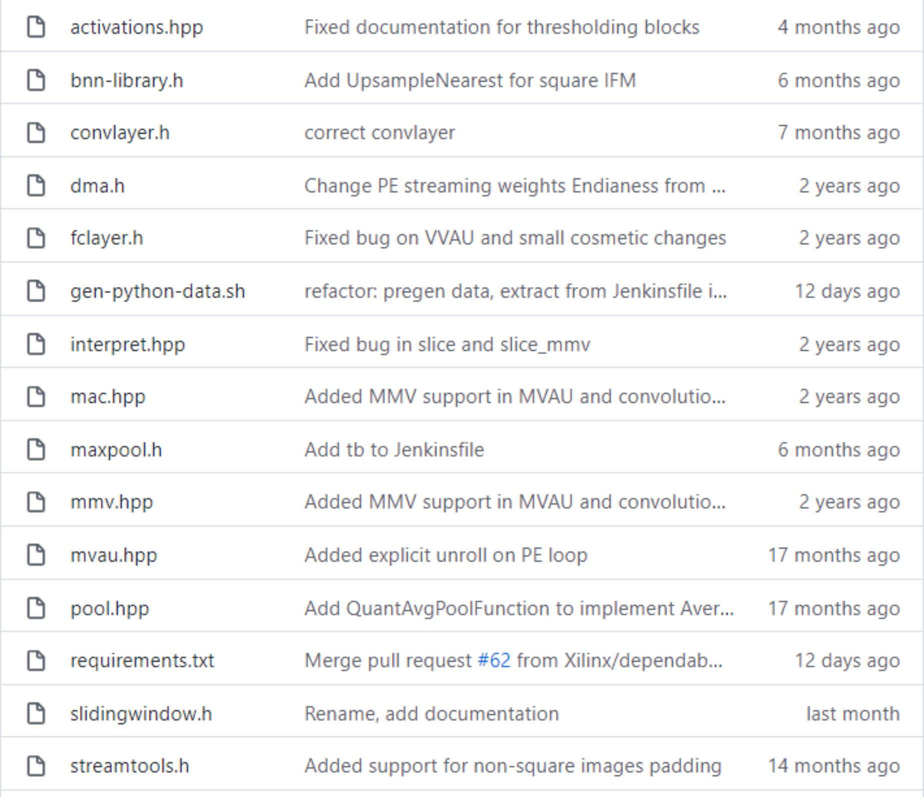
Customizable datatypes
Flexibility through C++ templates

Streaming I/O
Easily compose modules together, low latency

Customizable parallelism
In form of #PE and #SIMD
Control resource footprint & throughput

The FINN HLS Library

- An optimized, templated Vivado HLS C++ library of 10+ common DNN layers

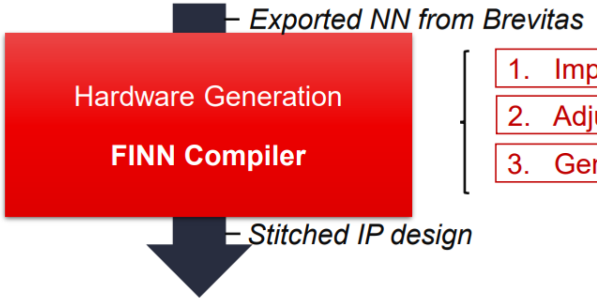


activations.hpp	Fixed documentation for thresholding blocks	4 months ago
bnn-library.h	Add UpsampleNearest for square IFM	6 months ago
convlayer.h	correct convlayer	7 months ago
dma.h	Change PE streaming weights Endianess from ...	2 years ago
fclayer.h	Fixed bug on VVAU and small cosmetic changes	2 years ago
gen-python-data.sh	refactor: pregen data, extract from Jenkinsfile i...	12 days ago
interpret.hpp	Fixed bug in slice and slice_mmv	2 years ago
mac.hpp	Added MMV support in MVAU and convolutio...	2 years ago
maxpool.h	Add tb to Jenkinsfile	6 months ago
mmv.hpp	Added MMV support in MVAU and convolutio...	2 years ago
mvau.hpp	Added explicit unroll on PE loop	17 months ago
pool.hpp	Add QuantAvgPoolFunction to implement Aver...	17 months ago
requirements.txt	Merge pull request #62 from Xilinx/dependab...	12 days ago
slidingwindow.h	Rename, add documentation	last month
streamtools.h	Added support for non-square images padding	14 months ago

More details next week!

https://github.com/Xilinx/finn-hlslib

FINN Compiler: Adjusting Performance/Resources



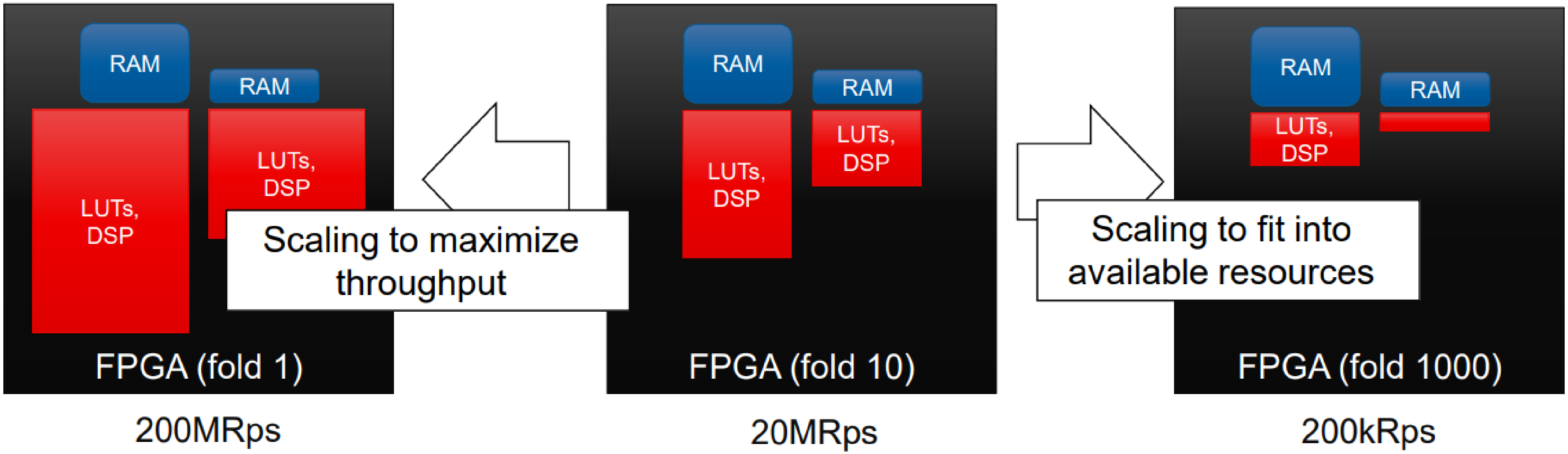
Exported NN from Brevitas

Hardware Generation

FINN Compiler

Stitched IP design

1. Import, streamlining transformations, conversion to HLS
2. Adjust folding to suit performance/resource requirements
3. Generate IP, and stitched IP design



Scaling to maximize throughput

Scaling to fit into available resources

FPGA (fold 1)

200MRps

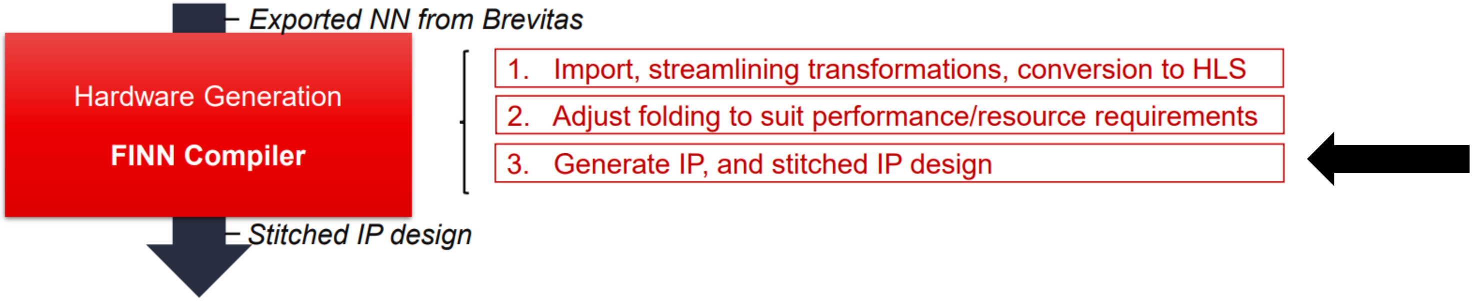
FPGA (fold 10)

20MRps

FPGA (fold 1000)

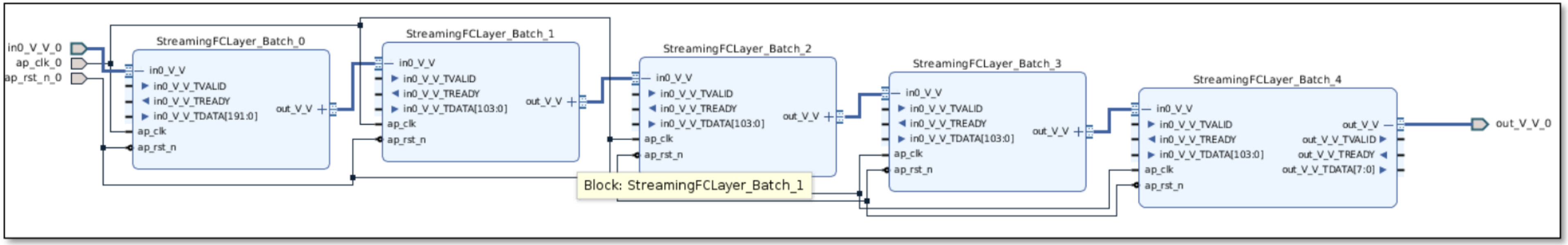
200kRps

FINN Compiler: IP Generation Flow

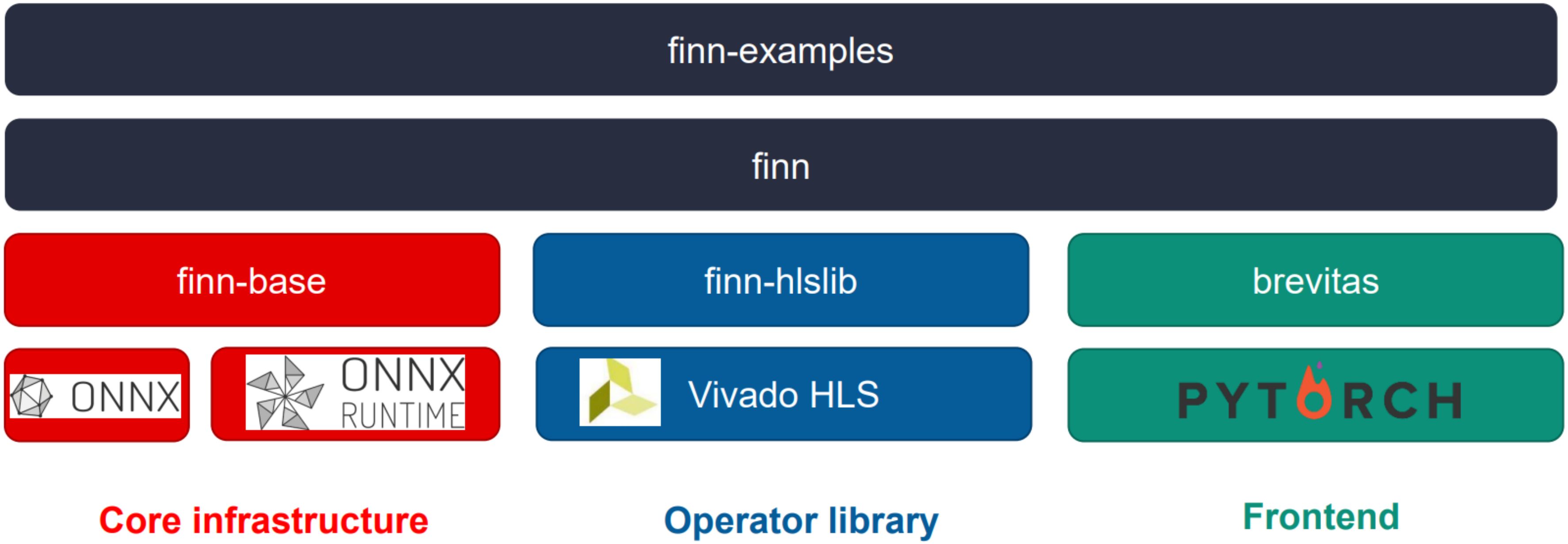


1. Import, streamlining transformations, conversion to HLS
2. Adjust folding to suit performance/resource requirements
3. Generate IP, and stitched IP design

- Stream-in, stream-out FPGA IP block
  - Easy "bump-in-the-wire" integration into streaming systems
  - Simple data movement, fully deterministic



Overview of the FINN software stack



FINN-Examples: Prebuilt Dataflow Accelerators

- Dataflow accelerators for MNIST, CIFAR-10, ImageNet
  - Bitfiles for PYNQ boards and Alveo U250
- Jupyter notebook example to run each accelerator
  - Based on PYNQ Python driver

```
# on your PYNQ board or Alveo U250
pip3 install finn-examples
pynq get-notebooks --from-package finn-examples -p .
```

- More examples
  - ResNet-50 toolflow (bitfiles already on Xilinx/ResNet50-PYNQ)
  - speech recognition & keyword spotting

GitHub https://github.com/Xilinx/finn-examples

FINN: Dataflow Compiler

- ONNX -> bitfile (or IPI design) build automation
- Docker environment with all dependencies
- Large library of graph transformations
  - Streamlining to remove floating point scaling factors
  - Lowering to finn-hlslib ops
  - Stitching generated IPs in Vivado IPI
- Custom ops corresponding to finn-hlslib
- Jupyter notebook tutorials (basic, advanced, end2end)

GitHub https://github.com/Xilinx/finn

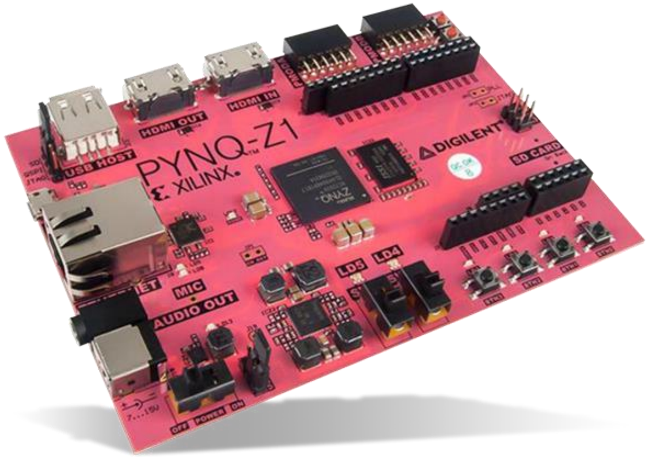
Brevitas: Quantization-Aware Training in PyTorch

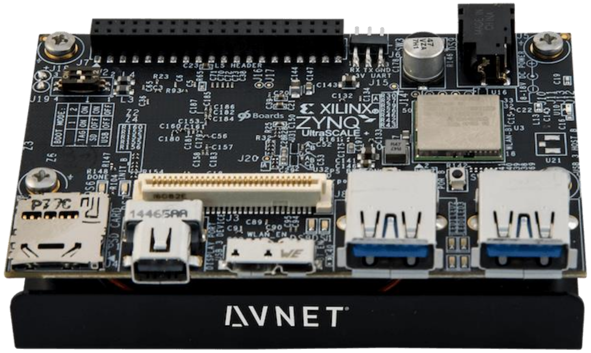
- Train NNs with quantized weights and activations
- Quantized versions of many PyTorch layers
  - e.g. *brevitas.nn.QuantConv2D* instead of *torch.nn.Conv2D*
- Flexible quantization schemes
  - Mixed precision with fixed or learnable bitwidths
- Example pretrained models + training scripts
  - image classification, speech-to-text, text-to-speech
- Different ONNX export flows for different backends in progress
  - FINN, Xilinx DPU, standard ONNX

GitHub **https://github.com/Xilinx/brevitas**

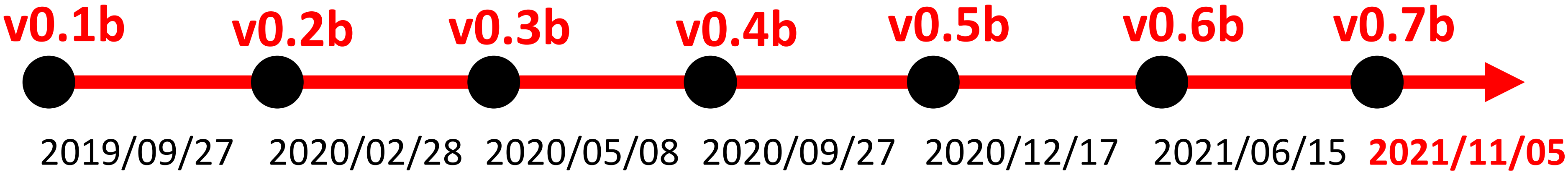
Supported Boards

- Edge: Pynq-Z1, Pynq-Z2, Ultra96 and ZCU104
- Datacenter: Alveo U250





FINN Release Verion



v0.1b 2019/09/27

v0.2b 2020/02/28

v0.3b 2020/05/08

v0.4b 2020/09/27

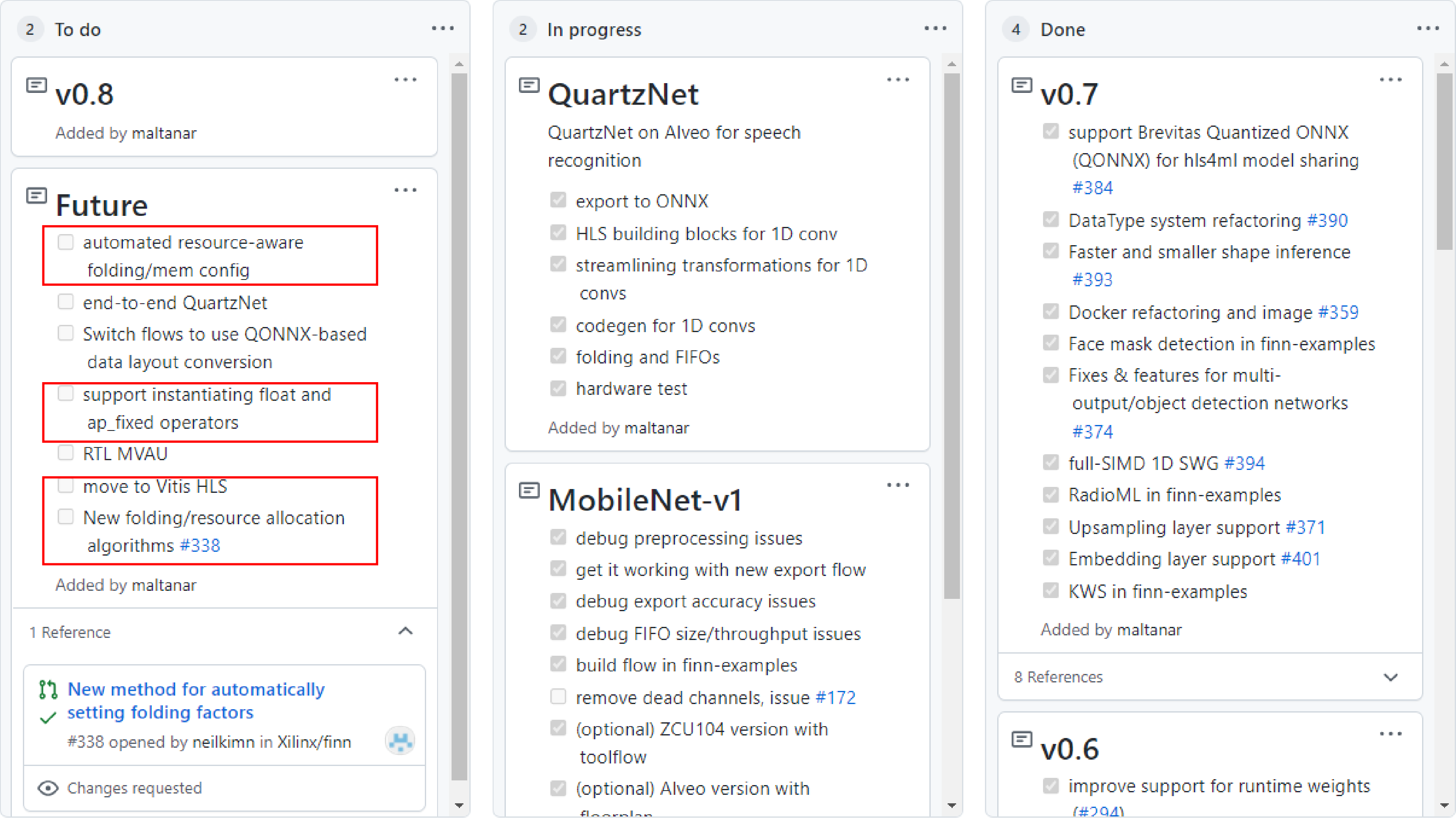
v0.5b 2020/12/17

v0.6b 2021/06/15

v0.7b **2021/11/05**

https://xilinx.github.io/finn/blog

FINN Road Map



2 To do

v0.8

Added by maltanar

Future

- □ automated resource-aware folding/mem config
- □ end-to-end QuartzNet
- □ Switch flows to use QONNX-based data layout conversion
- □ support instantiating float and ap_fixed operators
- □ RTL MVAU
- □ move to Vitis HLS
- □ New folding/resource allocation algorithms #338

Added by maltanar

1 Reference

New method for automatically setting folding factors
#338 opened by neilkimn in Xilinx/finn

Changes requested

2 In progress

QuartzNet

QuartzNet on Alveo for speech recognition

- ☑ export to ONNX
- ☑ HLS building blocks for 1D conv
- ☑ streamlining transformations for 1D convs
- ☑ codegen for 1D convs
- ☑ folding and FIFOs
- ☑ hardware test

Added by maltanar

MobileNet-v1

- ☑ debug preprocessing issues
- ☑ get it working with new export flow
- ☑ debug export accuracy issues
- ☑ debug FIFO size/throughput issues
- ☑ build flow in finn-examples
- □ remove dead channels, issue #172
- ☑ (optional) ZCU104 version with toolflow
- ☑ (optional) Alveo version with floorplan

4 Done

v0.7

- ☑ support Brevitas Quantized ONNX (QONNX) for hls4ml model sharing #384
- ☑ DataType system refactoring #390
- ☑ Faster and smaller shape inference #393
- ☑ Docker refactoring and image #359
- ☑ Face mask detection in finn-examples
- ☑ Fixes & features for multi-output/object detection networks #374
- ☑ full-SIMD 1D SWG #394
- ☑ RadioML in finn-examples
- ☑ Upsampling layer support #371
- ☑ Embedding layer support #401
- ☑ KWS in finn-examples

Added by maltanar

8 References

v0.6

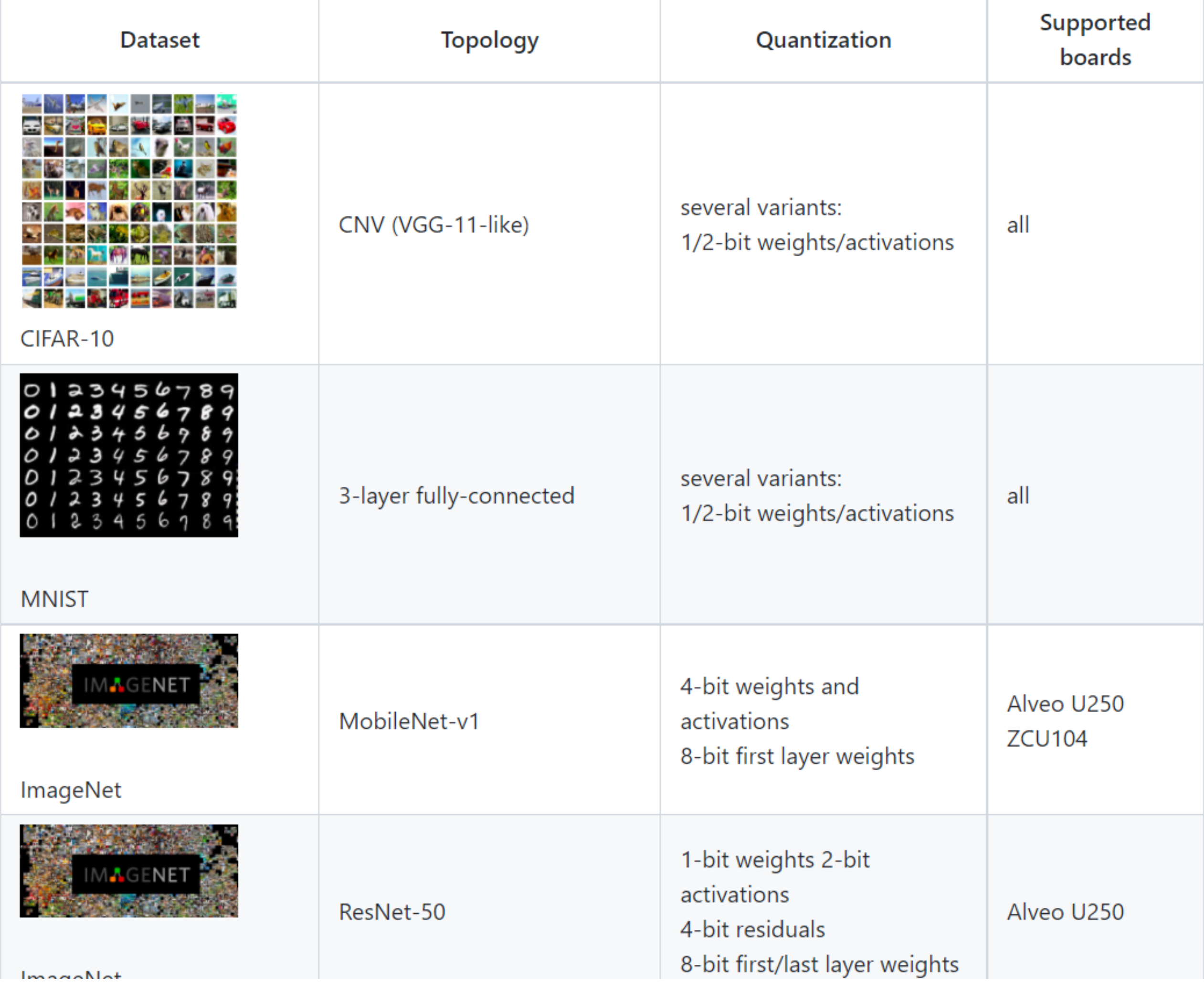
- ☑ improve support for runtime weights (#294)

FINN Application Example

- MNIST Handwritten Digit Classification
- Image Classification using CIFAR-10dataset
- Image Classification using ImageNet dataset
- Face mask wear and positioning
  - Low-power BNN classifier for Pynq-Z1 for correct face mask wear and positioning
- Radio signal modulation
  - Classify RadioML 2018.1 at 250k inferences/second on a ZCU104
- Keyword spotting
  - Trained on the Google Speech Commands v2 dataset

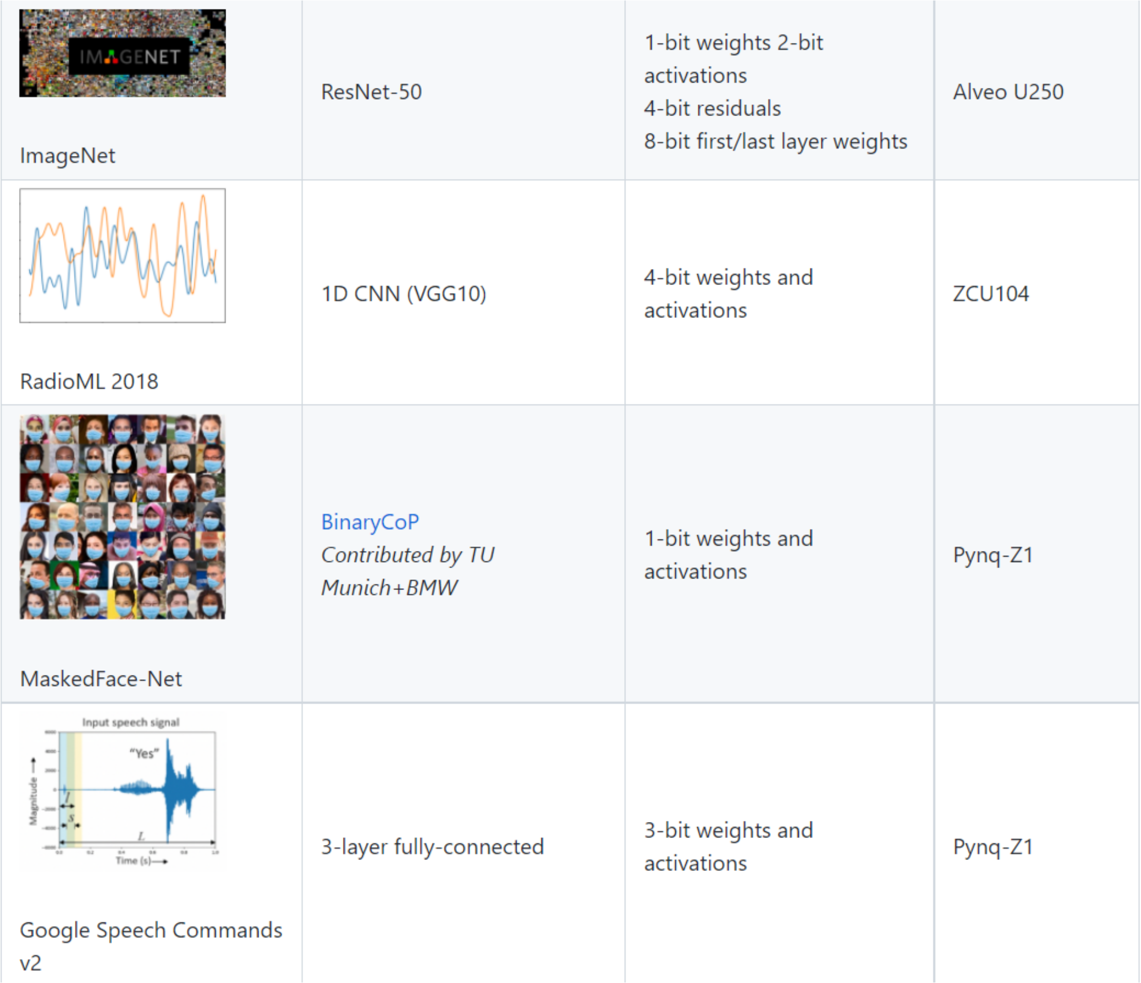
https://github.com/Xilinx/finn-examples/tree/main/finn_examples/notebooks

Example Neural Network Accelerators(1/2)



Dataset	Topology	Quantization	Supported boards
CIFAR-10	CNV (VGG-11-like)	several variants: 1/2-bit weights/activations	all
MNIST	3-layer fully-connected	several variants: 1/2-bit weights/activations	all
ImageNet	MobileNet-v1	4-bit weights and activations 8-bit first layer weights	Alveo U250 ZCU104
ImageNet	ResNet-50	1-bit weights 2-bit activations 4-bit residuals 8-bit first/last layer weights	Alveo U250

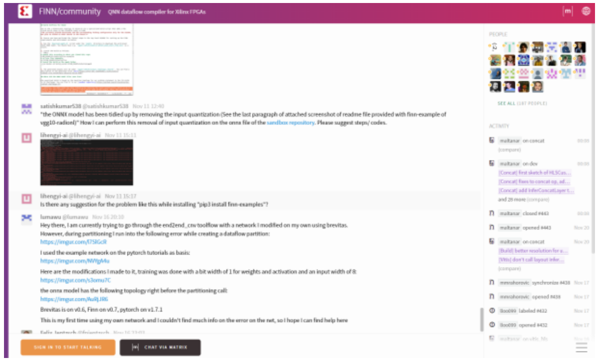
Example Neural Network Accelerators(2/2)



ImageNet	ResNet-50	1-bit weights 2-bit activations 4-bit residuals 8-bit first/last layer weights	Alveo U250
RadioML 2018	1D CNN (VGG10)	4-bit weights and activations	ZCU104
MaskedFace-Net	BinaryCoP *Contributed by TU Munich+BMW*	1-bit weights and activations	Pynq-Z1
Google Speech Commands v2	3-layer fully-connected	3-bit weights and activations	Pynq-Z1

FINN Community

- Old community :
  - **https://gitter.im/xilinx-finn/community**
- New community (Current) :
  - https://github.com/Xilinx/finn/discussions



FINN/community QNN dataflow compiler for Xilinx FPGAs

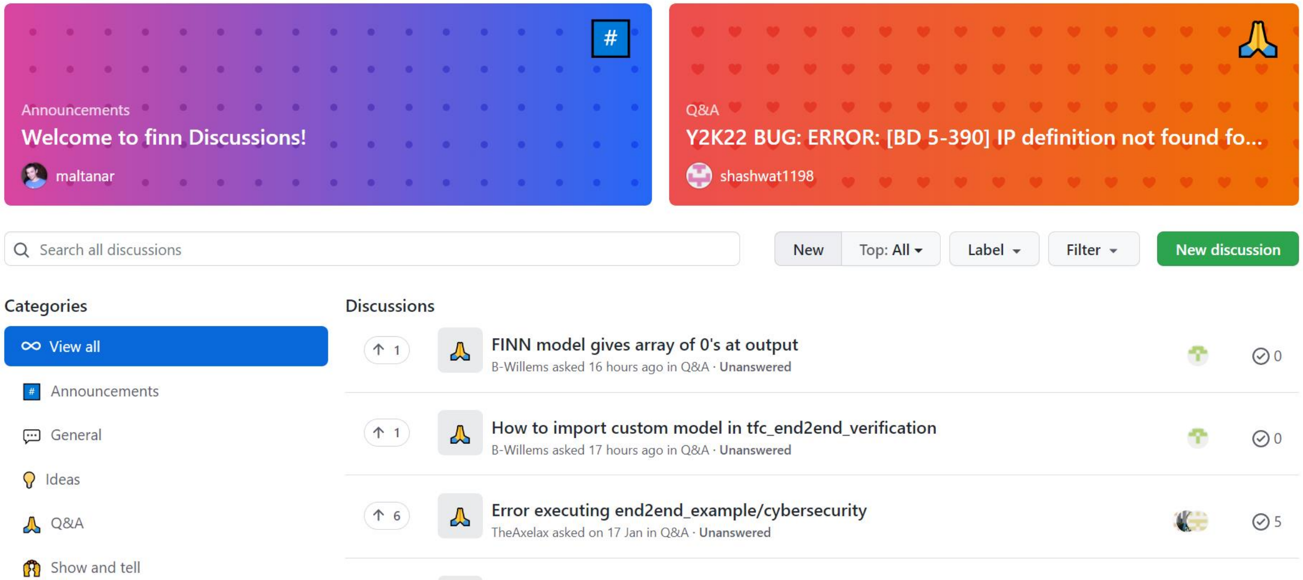
satishkumar538 @satishkumar538 Nov 11 12:40
"the ONNX model has been tidied up by removing the input quantization (See the last paragraph of attached screenshot of readme file provided with finn-example of vgg10-radioml)" How I can perform this removal of input quantization on the onnx file of the sandbox repository. Please suggest steps/ codes.

lihengyi-ai @lihengyi-ai Nov 11 15:11

lihengyi-ai @lihengyi-ai Nov 11 15:17
Is there any suggestion for the problem like this while installing "pip3 install finn-examples"?

lumawu @lumawu Nov 16 20:10
Hey there, I am currently trying to go through the end2end_cnv toolflow with a network I modified on my own using brevitas.
However, during partitioning I run into the following error while creating a dataflow partition:
https://imgur.com/f7SlGcR
I used the example network on the pytorch tutorials as basis:
https://imgur.com/NWYgM4u
Here are the modifications I made to it, training was done with a bit width of 1 for weights and activation and an input width of 8:
https://imgur.com/s3omu7C
the onnx model has the following topology right before the partitioning call:
https://imgur.com/AuRjJR6
Brevitas is on v0.6, Finn on v0.7, pytorch on v1.7.1
This is my first time using my own network and I couldn't find much info on the error on the net, so I hope I can find help here

Felix Jentzsch @fpjentzsch Nov 16 22:03



Announcements
Welcome to finn Discussions!
maltanar

Q&A
Y2K22 BUG: ERROR: [BD 5-390] IP definition not found fo...
shashwat1198

↑ 1 FINN model gives array of 0's at output
B-Willems asked 16 hours ago in Q&A · **Unanswered** 0

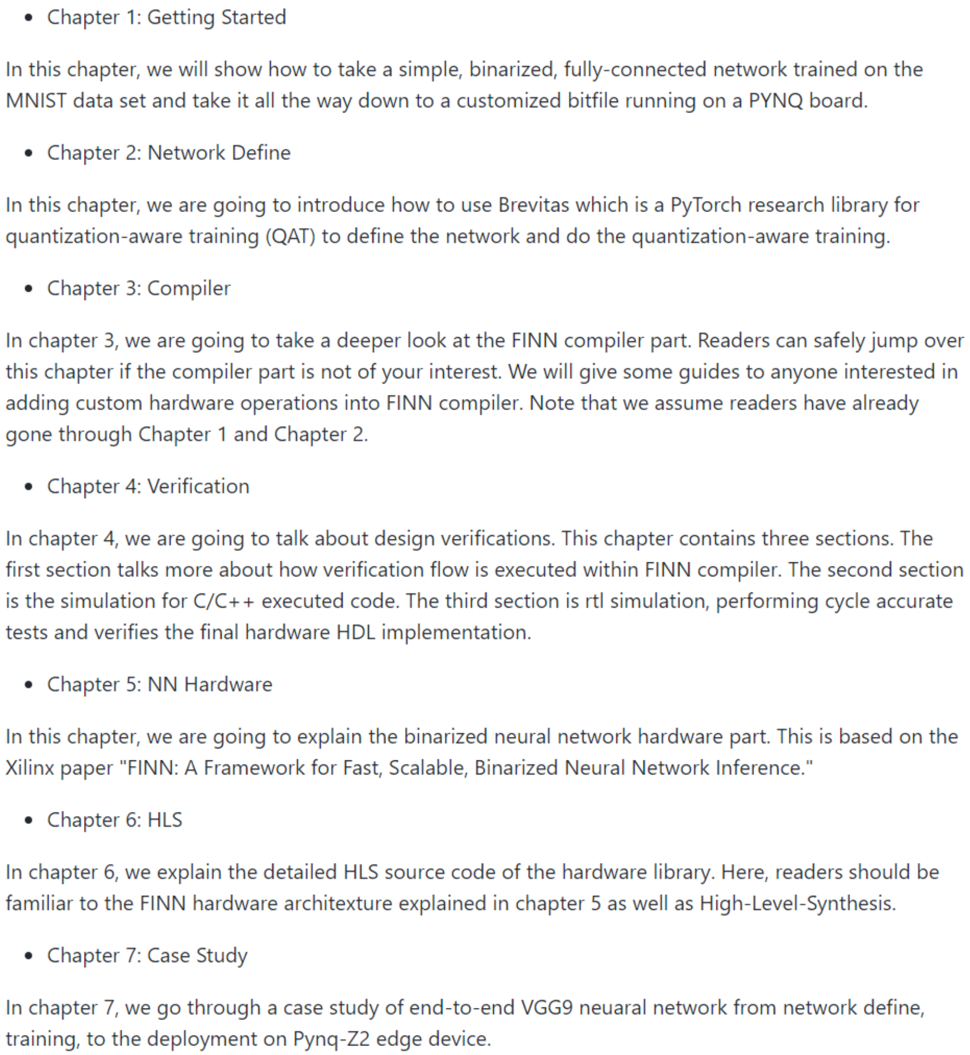
↑ 1 How to import custom model in tfc_end2end_verification
B-Willems asked 17 hours ago in Q&A · **Unanswered** 0

↑ 6 Error executing end2end_example/cybersecurity
TheAxelax asked on 17 Jan in Q&A · **Unanswered** 5

FINN Resource

- Xilinx Official Github Pages: https://xilinx.github.io/finn/
- Xilinx Official Documents: https://finn.readthedocs.io/en/latest/
- Xilinx FINN Repositories:
    - FINN framework: https://github.com/Xilinx/finn
    - finn-base: https://github.com/Xilinx/finn-base
    - finn-hls: https://github.com/Xilinx/finn-hlslib
- Xilinx Brevitas: https://github.com/Xilinx/brevitas
- Xilinx FINN Examples:
    - FINN Examples: https://github.com/Xilinx/finn-examples
    - BNN-PYNQ: https://github.com/Xilinx/BNN-PYNQ
- Xilinx FINN Publications: https://xilinx.github.io/finn/publications
    - FINN: A Framework for Fast, Scalable Binarized Neural Network Inference
    - FINN-R: An End-to-End Deep-Learning Framework for Fast Exploration of Quantized Neural Networks
- Xilinx Vitis HLS Pragmas: https://www.xilinx.com/html_docs/xilinx2021_1/vitis_doc/hls_pragmas.html

FINN Textbook



- Chapter 1: Getting Started

In this chapter, we will show how to take a simple, binarized, fully-connected network trained on the MNIST data set and take it all the way down to a customized bitfile running on a PYNQ board.

- Chapter 2: Network Define

In this chapter, we are going to introduce how to use Brevitas which is a PyTorch research library for quantization-aware training (QAT) to define the network and do the quantization-aware training.

- Chapter 3: Compiler

In chapter 3, we are going to take a deeper look at the FINN compiler part. Readers can safely jump over this chapter if the compiler part is not of your interest. We will give some guides to anyone interested in adding custom hardware operations into FINN compiler. Note that we assume readers have already gone through Chapter 1 and Chapter 2.

- Chapter 4: Verification

In chapter 4, we are going to talk about design verifications. This chapter contains three sections. The first section talks more about how verification flow is executed within FINN compiler. The second section is the simulation for C/C++ executed code. The third section is rtl simulation, performing cycle accurate tests and verifies the final hardware HDL implementation.

- Chapter 5: NN Hardware

In this chapter, we are going to explain the binarized neural network hardware part. This is based on the Xilinx paper "FINN: A Framework for Fast, Scalable, Binarized Neural Network Inference."

- Chapter 6: HLS

In chapter 6, we explain the detailed HLS source code of the hardware library. Here, readers should be familiar to the FINN hardware architexture explained in chapter 5 as well as High-Level-Synthesis.

- Chapter 7: Case Study

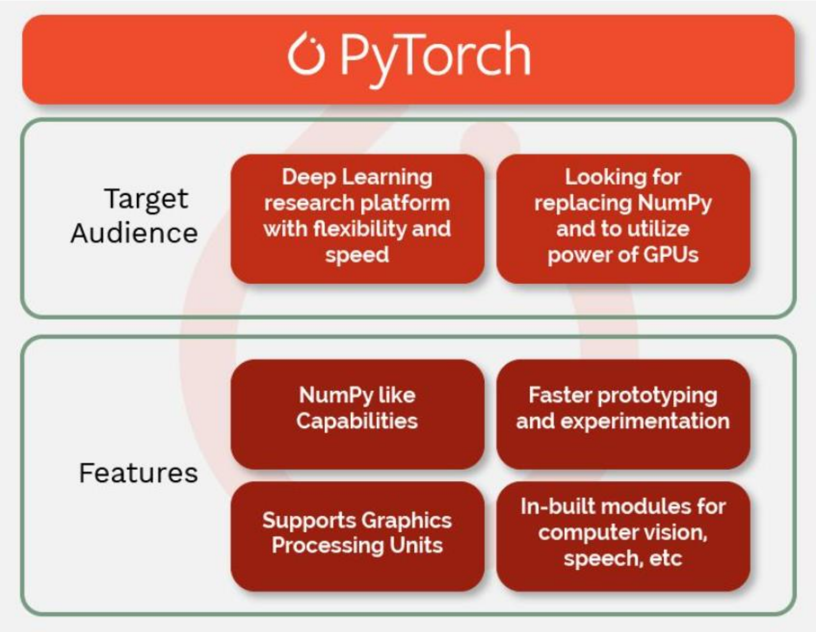
In chapter 7, we go through a case study of end-to-end VGG9 neuaral network from network define, training, to the deployment on Pynq-Z2 edge device.

Outline

- Introduction to FINN
- Network Define
- NN Hardware
- Lab Description

What is Pytorch

- PyTorch is a Python-based scientific computing package which uses the power of graphics processing units.
- It is considered as one of the best deep learning research platforms built to provide maximum flexibility and speed.



PyTorch

Target Audience

Deep Learning research platform with flexibility and speed

Looking for replacing NumPy and to utilize power of GPUs

Features

NumPy like Capabilities

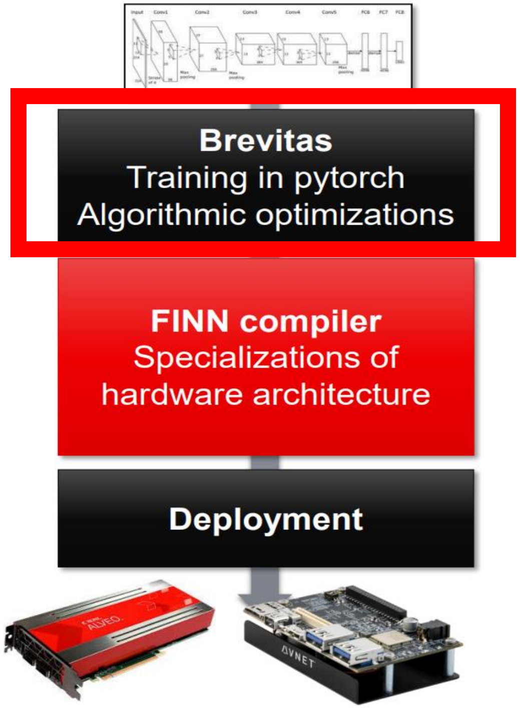
Faster prototyping and experimentation

Supports Graphics Processing Units

In-built modules for computer vision, speech, etc

What is Brevitas

- Brevitas is a PyTorch research library for quantization-aware training (QAT).
- Brevitas is targeting custom accelerators running on Xilinx FPGAs.



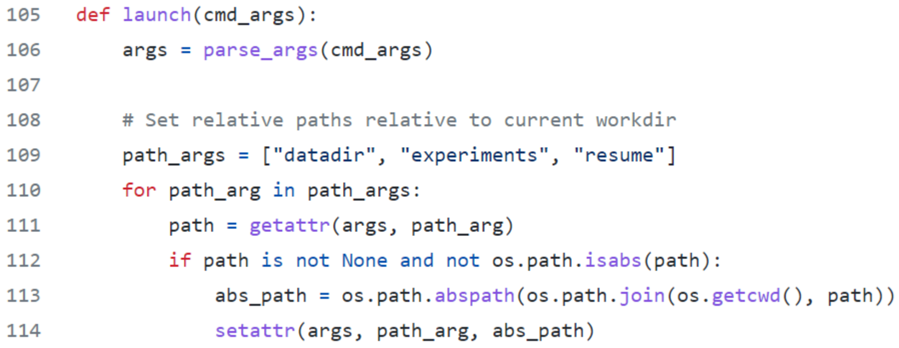
Brevitas
Training in pytorch
Algorithmic optimizations

FINN compiler
Specializations of
hardware architecture

Deployment

Brevitas Training Files

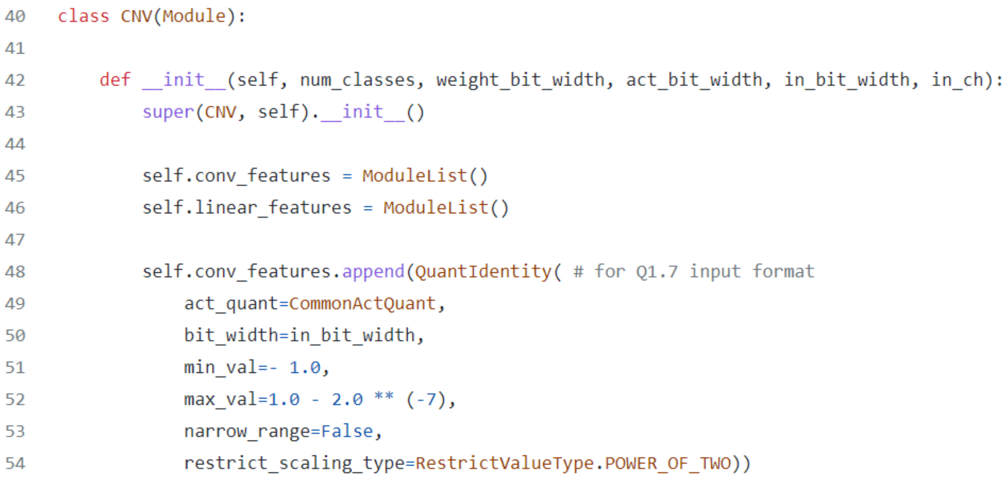
bnn_pynq_train.py



```python
def launch(cmd_args):
    args = parse_args(cmd_args)

    # Set relative paths relative to current workdir
    path_args = ["datadir", "experiments", "resume"]
    for path_arg in path_args:
        path = getattr(args, path_arg)
        if path is not None and not os.path.isabs(path):
            abs_path = os.path.abspath(os.path.join(os.getcwd(), path))
            setattr(args, path_arg, abs_path)
```

CNV.py



```python
class CNV(Module):

    def __init__(self, num_classes, weight_bit_width, act_bit_width, in_bit_width, in_ch):
        super(CNV, self).__init__()

        self.conv_features = ModuleList()
        self.linear_features = ModuleList()

        self.conv_features.append(QuantIdentity( # for Q1.7 input format
            act_quant=CommonActQuant,
            bit_width=in_bit_width,
            min_val=- 1.0,
            max_val=1.0 - 2.0 ** (-7),
            narrow_range=False,
            restrict_scaling_type=RestrictValueType.POWER_OF_TWO))
```

Fully-Connected Layer

```
QuantLinear(
    in_features=in_features,
    out_features=out_features,
    bias=False,
    weight_bit_width=weight_bit_width,
    weight_quant=CommonWeightQuant)
)
```

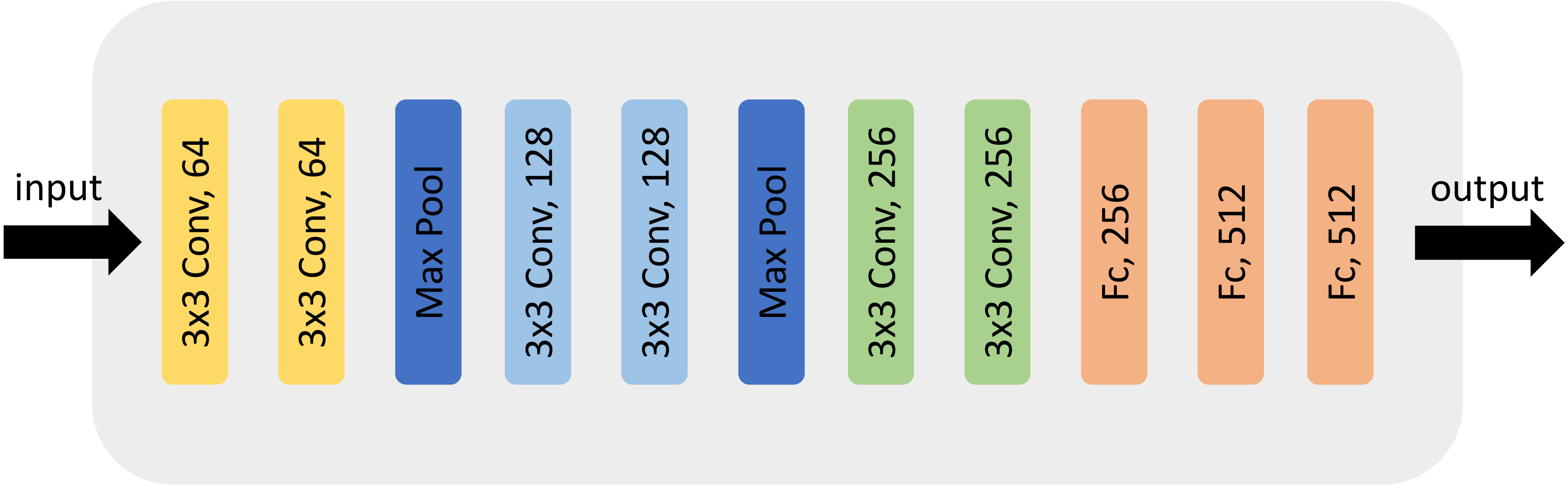
Convolution Layer

```
QuantConv2d(
    kernel_size=KERNEL_SIZE,
    in_channels=in_ch,
    out_channels=out_ch,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width)
)
```

Other commonly used functions

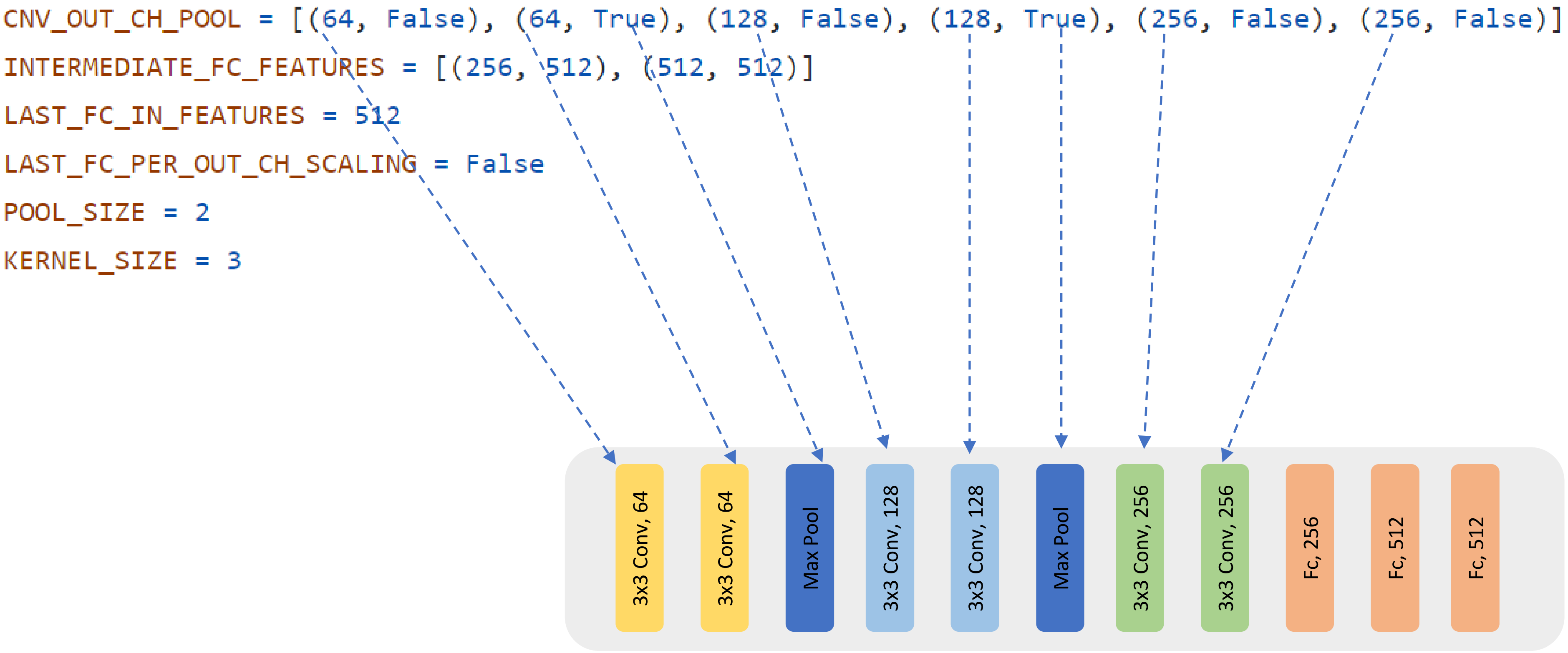
- Batch Normalization
  - **BatchNorm1d(num_features=in_features)**
  - **BatchNorm2d(in_ch, eps= eps)**
- Max Pooling
  - **MaxPool2d(kernel_size= kernel_size)**
- Dropout
  - **Dropout(p=DROPOUT)**

Example


input
3x3 Conv, 64
3x3 Conv, 64
Max Pool
3x3 Conv, 128
3x3 Conv, 128
Max Pool
3x3 Conv, 256
3x3 Conv, 256
Fc, 256
Fc, 512
Fc, 512
output

Network Define(1/)

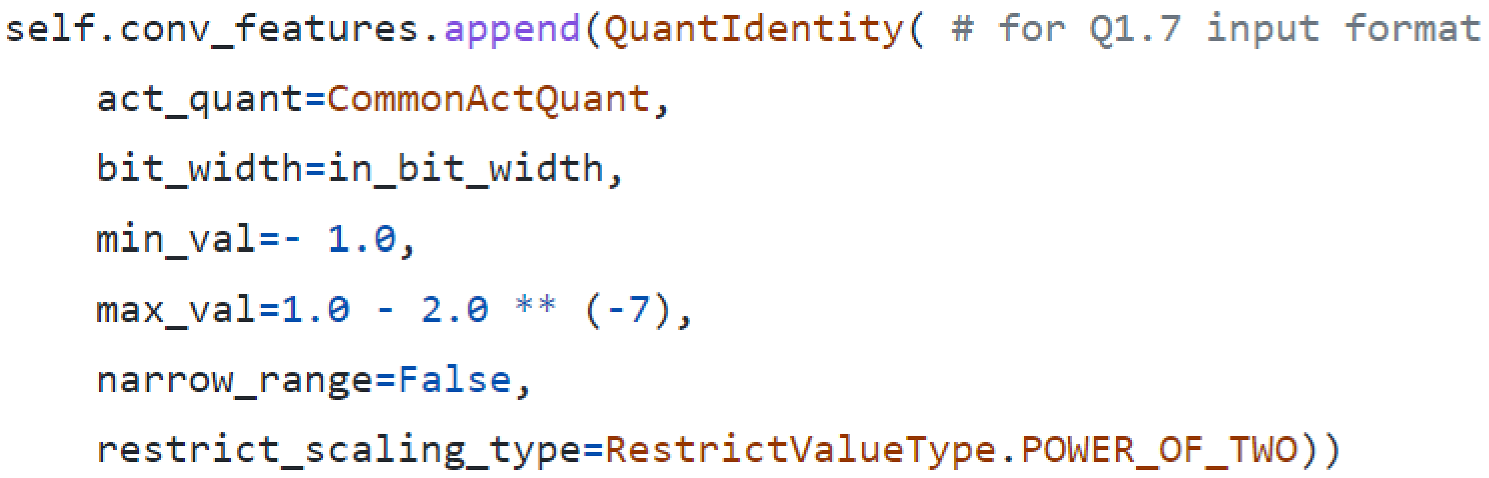
- Define the network architecture



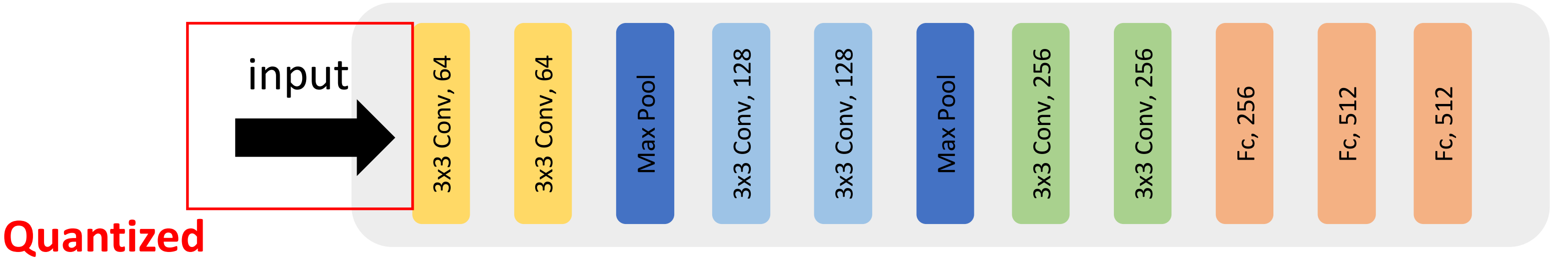
```
CNV_OUT_CH_POOL = [(64, False), (64, True), (128, False), (128, True), (256, False), (256, False)]
INTERMEDIATE_FC_FEATURES = [(256, 512), (512, 512)]
LAST_FC_IN_FEATURES = 512
LAST_FC_PER_OUT_CH_SCALING = False
POOL_SIZE = 2
KERNEL_SIZE = 3
```

Network Define(2/)

- Quantized the input data



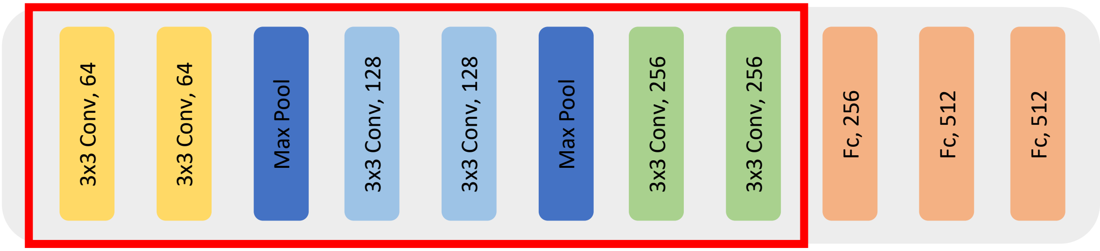
```
self.conv_features.append(QuantIdentity( # for Q1.7 input format
    act_quant=CommonActQuant,
    bit_width=in_bit_width,
    min_val=- 1.0,
    max_val=1.0 - 2.0 ** (-7),
    narrow_range=False,
    restrict_scaling_type=RestrictValueType.POWER_OF_TWO))
```



Network Define(3/)

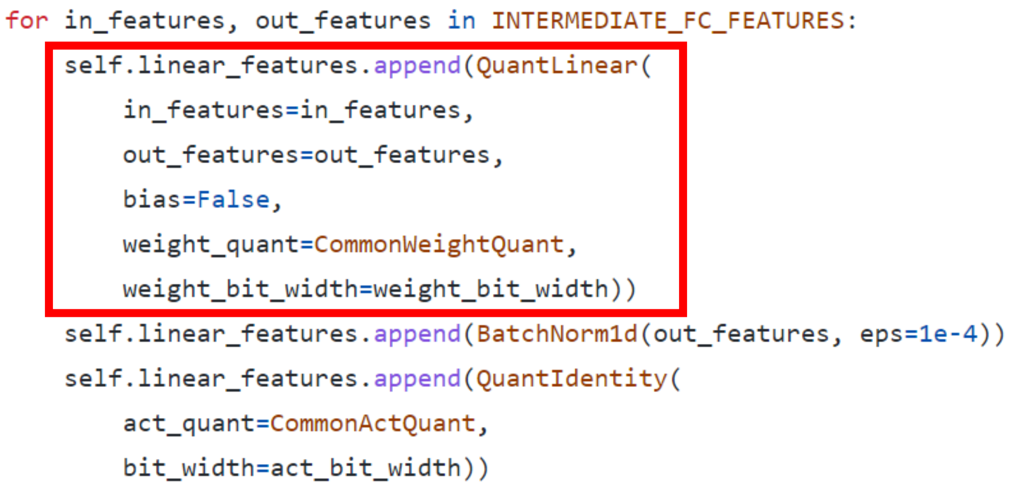
- Append the conv and Maxpool layers into our network

```
for out_ch, is_pool_enabled in CNV_OUT_CH_POOL:
    self.conv_features.append(QuantConv2d(
        kernel_size=KERNEL_SIZE,
        in_channels=in_ch,
        out_channels=out_ch,
        bias=False,
        weight_quant=CommonWeightQuant,
        weight_bit_width=weight_bit_width))
    in_ch = out_ch
    self.conv_features.append(BatchNorm2d(in_ch, eps=1e-4))
    self.conv_features.append(QuantIdentity(
        act_quant=CommonActQuant,
        bit_width=act_bit_width))
    if is_pool_enabled:
        self.conv_features.append(MaxPool2d(kernel_size=2))
```

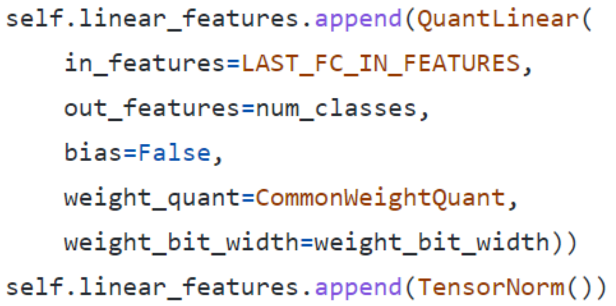


Network Define(4/)

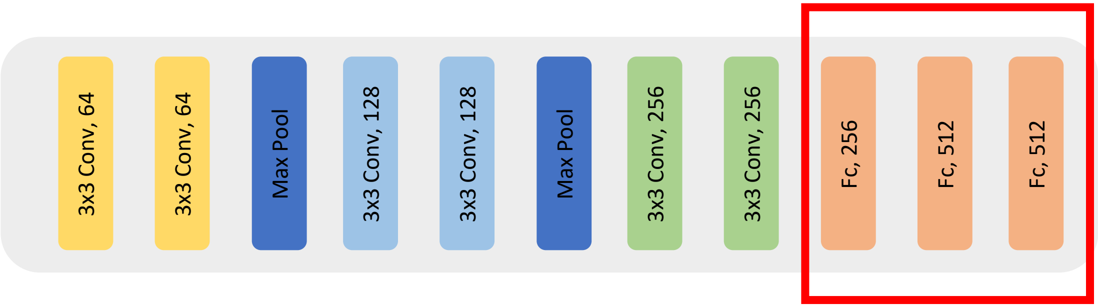
- Append the Fully Connected layers into our network



```
for in_features, out_features in INTERMEDIATE_FC_FEATURES:
    self.linear_features.append(QuantLinear(
        in_features=in_features,
        out_features=out_features,
        bias=False,
        weight_quant=CommonWeightQuant,
        weight_bit_width=weight_bit_width))
    self.linear_features.append(BatchNorm1d(out_features, eps=1e-4))
    self.linear_features.append(QuantIdentity(
        act_quant=CommonActQuant,
        bit_width=act_bit_width))
```



```
self.linear_features.append(QuantLinear(
    in_features=LAST_FC_IN_FEATURES,
    out_features=num_classes,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))
self.linear_features.append(TensorNorm())
```



Outline

- Introduction to FINN
- Network Define
- NN Hardware
- Lab Description

Outline

- Introduction to FINN
- Network Define
- NN Hardware
- Lab Description