Lab-FINN

Fast, Scalable Quantized Neural Network Inference on FPGAs, FINN

BOL-EDU

2022 Spring

©BOLEDU

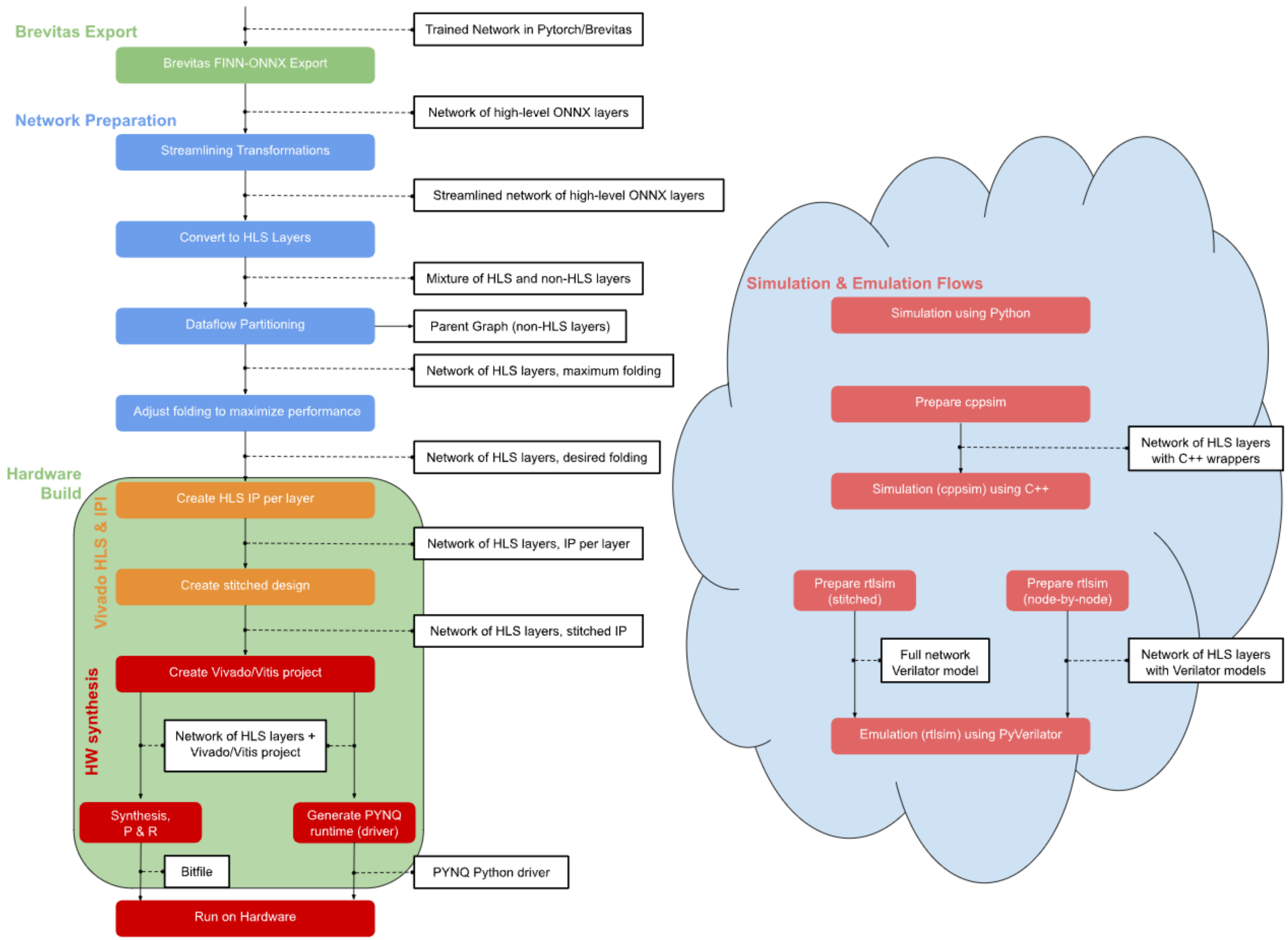
Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance Improvement
- Questions
- Report & Submission

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
- Report & Submission

Quick Recap of the End-to-End Flow



©BOLEDU

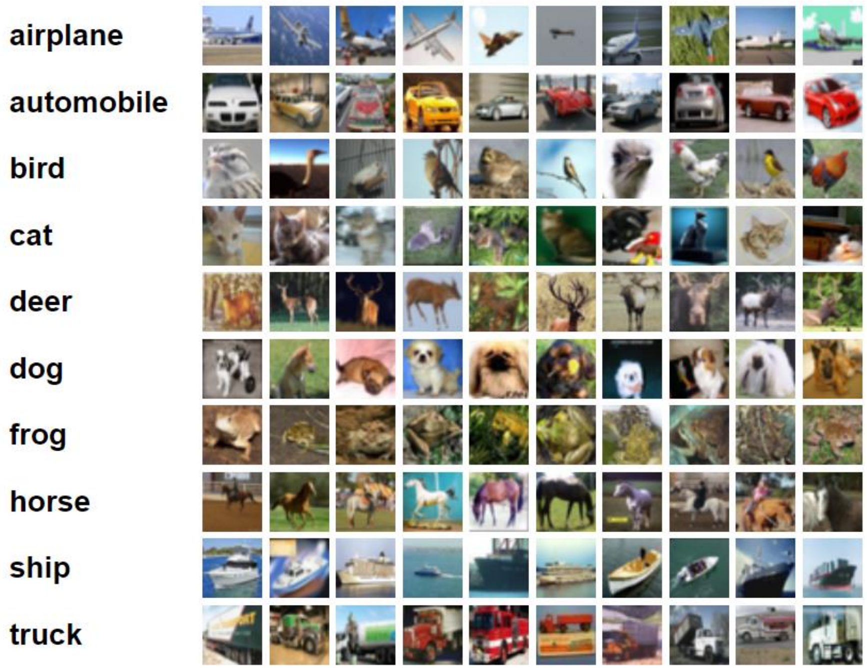
Quick Introduction to the CNV-w1a1 Network

- Input size: 32x32 8bits RGB Data
- Weights and Activations are binary values.
- Output is a number that represents the result predicted by our model.

©BOLEDU

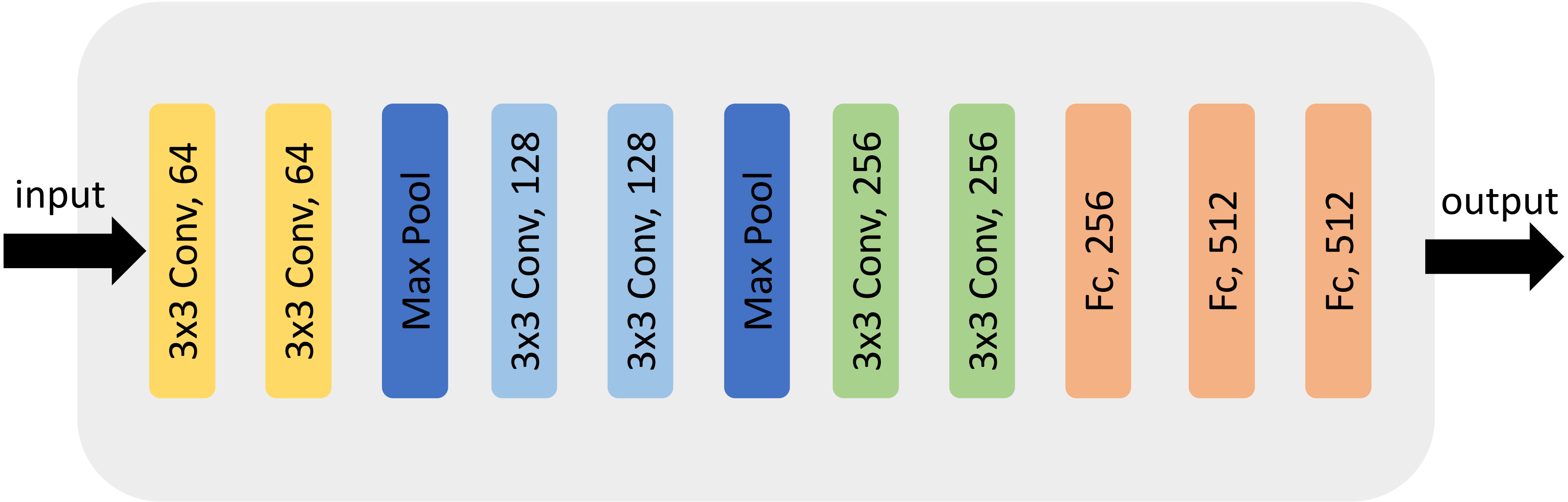
CIFAR-10 dataset

- The CIFAR-10 dataset consists of 60000 32x32 colour images in 10 classes, with 6000 images per class.
- There are 50000 training images and 10000 test images.



©BOLEDU

Network Architecture



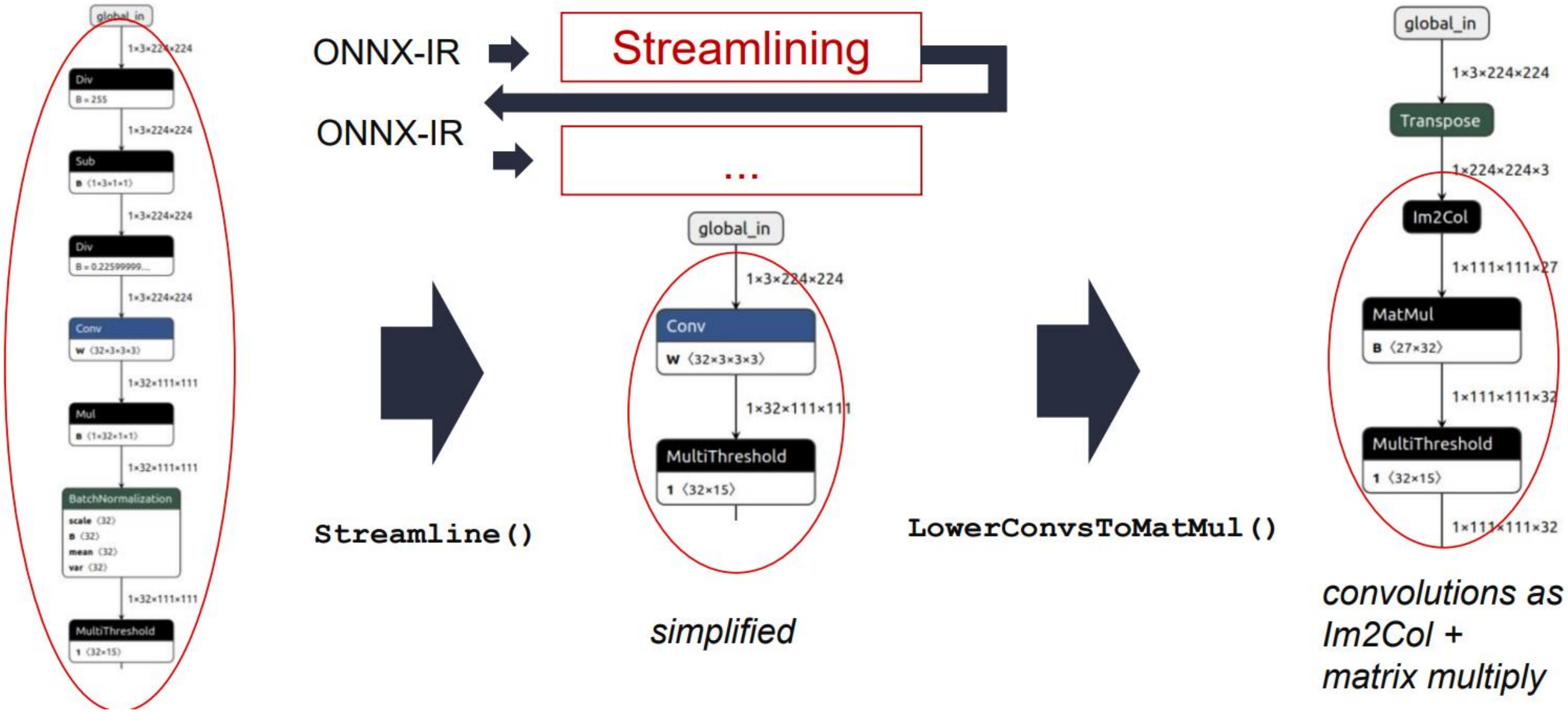
Brevitas Export, FINN Import and Tidy-Up

- We will start by exporting the pretrained CNV-w1a1 network to ONNX.
- Run the "tidy-up" transformations to have a first look at the topology.

```
import onnx
from finn.util.test import get_test_model_trained
import brevitas.onnx as bo
from finn.core.modelwrapper import ModelWrapper
from finn.transformation.infer_shapes import InferShapes
from finn.transformation.fold_constants import FoldConstants
from finn.transformation.general import GiveReadableTensorNames, GiveUniqueNodeNames, RemoveStaticGraphInputs

cnv = get_test_model_trained("CNV", 1, 1)
bo.export_finn_onnx(cnv, (1, 3, 32, 32), build_dir + "/end2end_cnv_w1a1_export.onnx")
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_export.onnx")
model = model.transform(InferShapes())
model = model.transform(FoldConstants())
model = model.transform(GiveUniqueNodeNames())
model = model.transform(GiveReadableTensorNames())
model = model.transform(RemoveStaticGraphInputs())
model.save(build_dir + "/end2end_cnv_w1a1_tidy.onnx")
```

©BOLEDU

Transformation



Transformation

- Pre-processing
  - Divides the input uint8 data by 255 so the inputs to the CNV-w1a1 network are bounded between [0, 1]
- Post-processing
  - Takes the output of the network and returns the index (0-9) of the image category with the highest probability (top-1).

©BOLEDU

Dataflow Partitioning

- We'll first convert the layers that we can put into the FPGA into their HLS equivalents and separate them out into a dataflow partition.

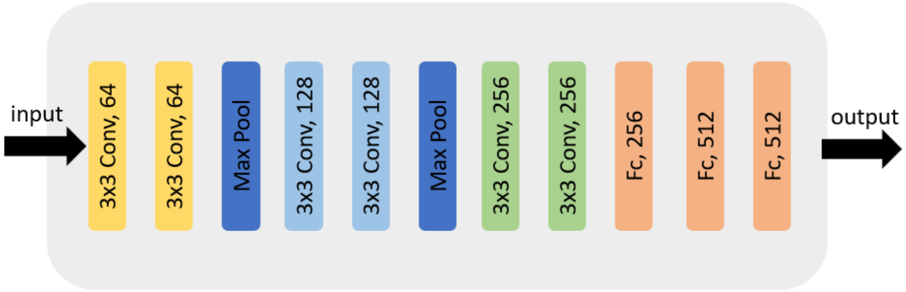
```
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_streamlined.onnx")
model = model.transform(to_hls.InferBinaryStreamingFCLayer(mem_mode))
model = model.transform(to_hls.InferQuantizedStreamingFCLayer(mem_mode))
# TopK to LabelSelect
model = model.transform(to_hls.InferLabelSelectLayer())
# input quantization (if any) to standalone thresholding
model = model.transform(to_hls.InferThresholdingLayer())
model = model.transform(to_hls.InferConvInpGen())
model = model.transform(to_hls.InferStreamingMaxPool())
# get rid of Reshape(-1, 1) operation between hlslib nodes
model = model.transform(RemoveCNVtoFCFlatten())
# get rid of Tranpose -> Tranpose identity seq
model = model.transform(absorb.AbsorbConsecutiveTransposes())
# infer tensor data layouts
model = model.transform(InferDataLayouts())
parent_model = model.transform(CreateDataflowPartition())
parent_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_parent.onnx")
sdp_node = parent_model.get_nodes_by_op_type("StreamingDataflowPartition")[0]
sdp_node = getCustomOp(sdp_node)
dataflow_model_filename = sdp_node.get_nodeattr("model")
# save the dataflow partition with a different name for easier access
dataflow_model = ModelWrapper(dataflow_model_filename)
dataflow_model.save(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
```

©BOLEDU

Adjust folding factor

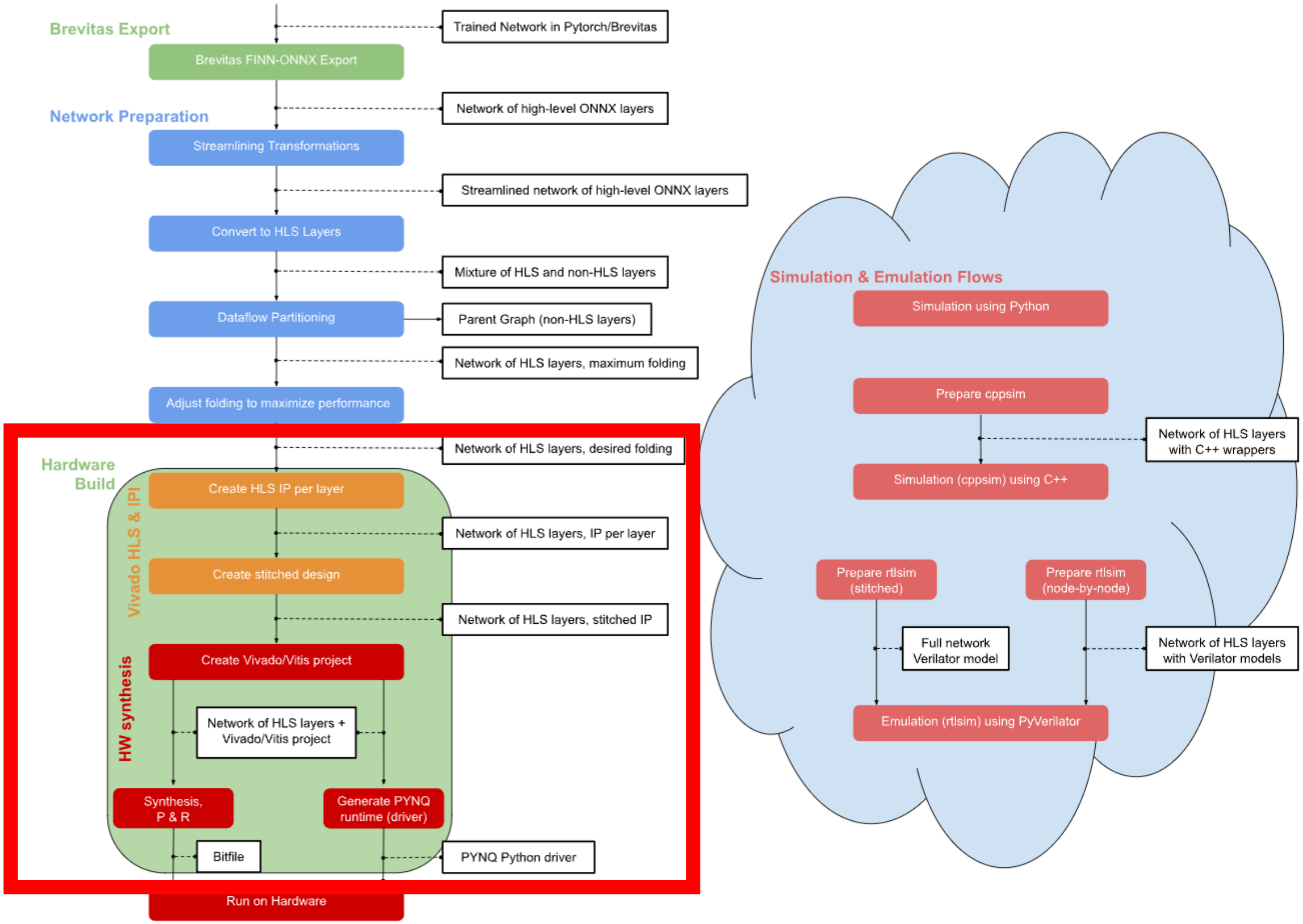
- We have to set the folding factors for certain layers to adjust the performance of our accelerator.

```
model = ModelWrapper(build_dir + "/end2end_cnv_w1a1_dataflow_model.onnx")
fc_layers = model.get_nodes_by_op_type("StreamingFCLayer_Batch")
# each tuple is (PE, SIMD, in_fifo_depth) for a layer
folding = [
    (16, 3, 128),
    (32, 32, 128),
    (16, 32, 128),
    (16, 32, 128),
    (4, 32, 81),
    (1, 32, 2),
    (1, 4, 2),
    (1, 8, 128),
    (5, 1, 3),
]
for fcl, (pe, simd, ififodepth) in zip(fc_layers, folding):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFODepth", ififodepth)

# use same SIMD values for the sliding window operators
swg_layers = model.get_nodes_by_op_type("ConvolutionInputGenerator")
for i in range(len(swg_layers)):
    swg_inst = getCustomOp(swg_layers[i])
    simd = folding[i][1]
    swg_inst.set_nodeattr("SIMD", simd)

model = model.transform(GiveUniqueNodeNames())
model.save(build_dir + "/end2end_cnv_w1a1_folded.onnx")
```



©BOLEDU

Hardware Build



©BOLEDU

Hardware Generation

- Specify the target board and clock period
- ZynqBuild is the hardware generation function
- This step may take about 120 minutes depending on your host computer.

```
test_pynq_board = "Pynq-Z2"
target_clk_ns = 10

from finn.transformation.fpgadataflow.make_zynq_proj import ZynqBuild
model = ModelWrapper(build_dir+"/end2end_cnv_w1a1_folded.onnx")
model = model.transform(ZynqBuild(platform = test_pynq_board, period_ns = target_clk_ns))
model.save(build_dir + "/end2end_cnv_w1a1_synth.onnx")
```

©BOLEDU

Deployment and Remote Execution

- Set up our pynq-board's IP and PORT
  - For the PYNQ IPs and accounts, please consult your TA

```
import os

# set up the following values according to your own environment
# FINN will use ssh to deploy and run the generated accelerator
ip = os.getenv("PYNQ_IP", "192.168.2.99")
username = os.getenv("PYNQ_USERNAME", "xilinx")
password = os.getenv("PYNQ_PASSWORD", "xilinx")
port = os.getenv("PYNQ_PORT", 22)
target_dir = os.getenv("PYNQ_TARGET_DIR", "/home/xilinx/finn_cnv_end2end_example")
# set up ssh options to only allow publickey authentication
options = "-o PreferredAuthentications=publickey -o PasswordAuthentication=no"

# test access to PYNQ board
! ssh {options} {username}@{ip} -p {port} cat /var/run/motd.dynamic
```


©BOLEDU

Download the dataset

- Connect to PYNQ-board and download the dataset

```
! ssh {options} -t {username}@{ip} -p {port}
'echo {password} | sudo -S pip3 install git+https://github.com/fbcotter/dataset_loading.git@0.0.4#egg=dataset_loading'
```

- If your pynq cannot download the mnist dataset, then post the issue to the LAB-FINN discussion space

©BOLEDU

Validating the Accuracy on a PYNQ Board

- Connect to PYNQ-board
- Validating the Accuracy

```
! ssh {options} -t {username}@{ip} -p {port}
'cd {target_dir_pynq}; echo {password} | sudo -S python3.6 validate.py --dataset cifar10 --batchsize 1000'
```

```
[sudo] password for xilinx: Tar File found in dest_dir. Not Downloading again
Extracting Python CIFAR10 data.
Files extracted
batch 1 / 10 : total OK 851 NOK 149
batch 2 / 10 : total OK 1683 NOK 317
batch 3 / 10 : total OK 2522 NOK 478
batch 4 / 10 : total OK 3370 NOK 630
batch 5 / 10 : total OK 4207 NOK 793
batch 6 / 10 : total OK 5044 NOK 956
batch 7 / 10 : total OK 5887 NOK 1113
batch 8 / 10 : total OK 6728 NOK 1272
batch 9 / 10 : total OK 7570 NOK 1430
batch 10 / 10 : total OK 8419 NOK 1581
Final accuracy: 84.190000
```

Screen Dump this result

©BOLEDU

Part1 Requirements

- Screen Dump
  - Each ONNX Graph
  - Model Accuracy
- Describe your observations and understanding of each transformation

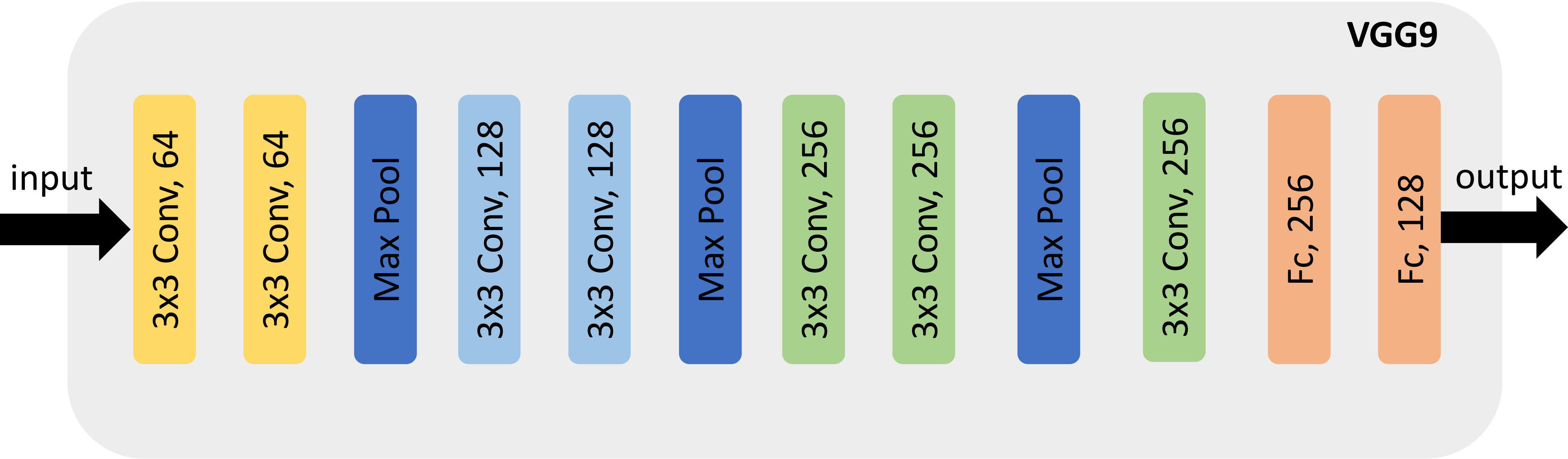
©BOLEDU

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
- Report & Submission

©BOLEDU

VGG9


VGG9
input
3x3 Conv, 64
3x3 Conv, 64
Max Pool
3x3 Conv, 128
3x3 Conv, 128
Max Pool
3x3 Conv, 256
3x3 Conv, 256
Max Pool
3x3 Conv, 256
Fc, 256
Fc, 128
output

©BOLEDU

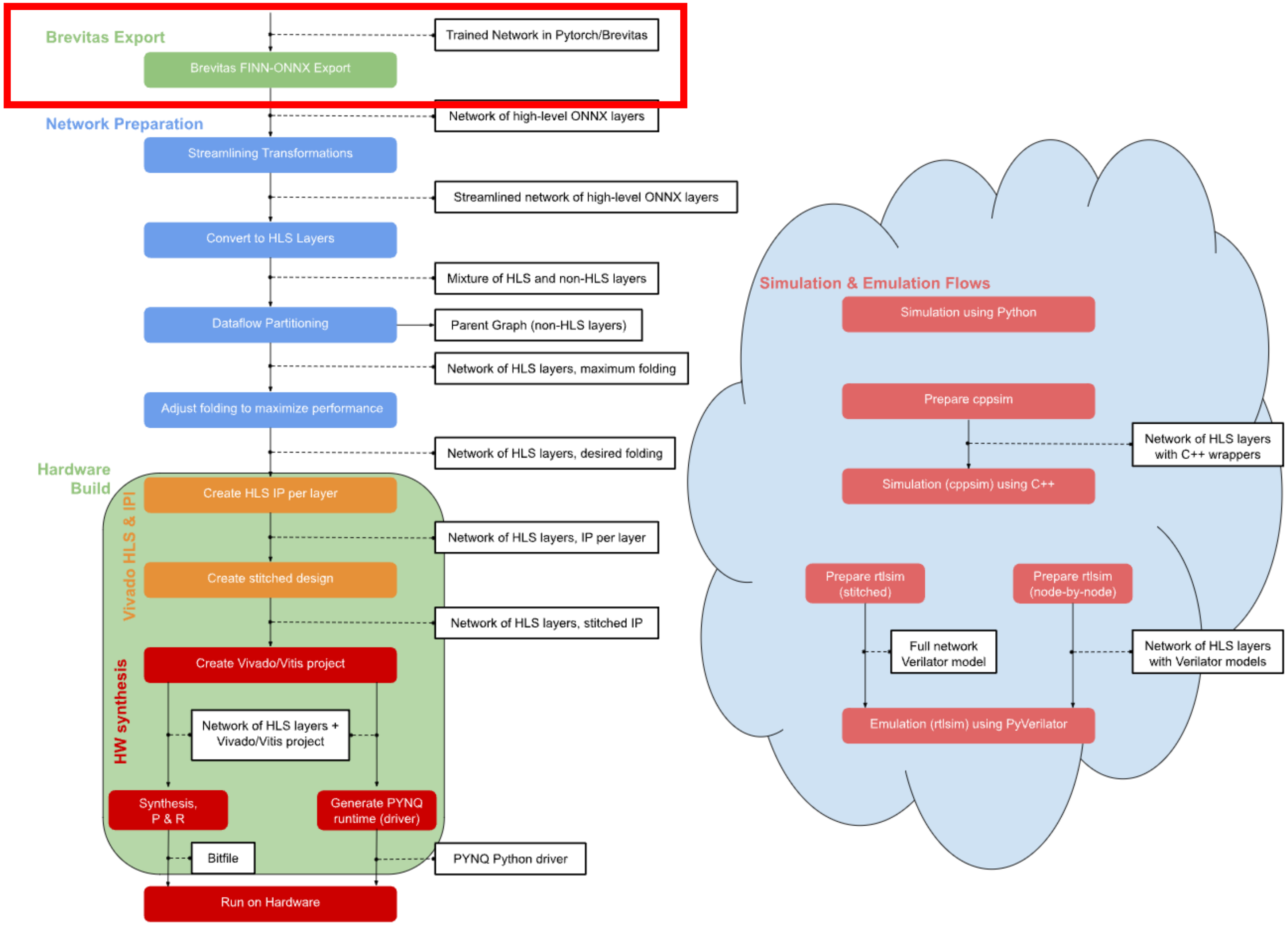
CIFAR100

- It has 100 classes containing 600 images each.
- There are 500 training images and 100 testing images per class.

Classes
beaver, dolphin, otter, seal, whale
aquarium fish, flatfish, ray, shark, trout
orchids, poppies, roses, sunflowers, tulips
bottles, bowls, cans, cups, plates
apples, mushrooms, oranges, pears, sweet peppers
clock, computer keyboard, lamp, telephone, television
bed, chair, couch, table, wardrobe
bee, beetle, butterfly, caterpillar, cockroach
bear, leopard, lion, tiger, wolf
bridge, castle, house, road, skyscraper
cloud, forest, mountain, plain, sea
camel, cattle, chimpanzee, elephant, kangaroo
fox, porcupine, possum, raccoon, skunk
crab, lobster, snail, spider, worm
baby, boy, girl, man, woman
crocodile, dinosaur, lizard, snake, turtle
hamster, mouse, rabbit, shrew, squirrel
maple, oak, palm, pine, willow
bicycle, bus, motorcycle, pickup truck, train
lawn-mower, rocket, streetcar, tank, tractor

https://www.cs.toronto.edu/~kriz/cifar.html

Model Prepare



©BOLEDU

Define Network Architecture

- We need to define our network architecture in CNV.py

Change the following simple Network example to VGG9

```
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
```

CNV.py

©BOLEDU

Specify Dataset

- Change Dataset from CIFAR-10 to CIFAR-100

```python
if dataset == 'CIFAR10':
    train_transforms_list = [transforms.RandomCrop(32, padding=4),
                             transforms.RandomHorizontalFlip(),
                             transforms.ToTensor()]
    transform_train = transforms.Compose(train_transforms_list)
    builder = CIFAR10

elif dataset == 'MNIST':
    transform_train = transform_to_tensor
    builder = MirrorMNIST
else:
    raise Exception("Dataset not supported: {}".format(args.dataset))
```

bnn_pynq_train.py

©BOLEDU

Configuration File

- Create a configuration file for VGG9
- Bit width of weigh and activation is 1. (Binary Network)

```
[MODEL]
ARCH: CNV
PRETRAINED_URL: https://github.com/Xilinx/brevitas/releases/download/bnn_pynq-r0/cnv_1w1a-758c8fef.pth
EVAL_LOG: https://github.com/Xilinx/brevitas/releases/download/cnv_test_ref-r0/cnv_1w1a_eval-ea2f0427.txt
DATASET: CIFAR10
IN_CHANNELS: 3
NUM_CLASSES: 10

[QUANT]
WEIGHT_BIT_WIDTH: 1
ACT_BIT_WIDTH: 1
IN_BIT_WIDTH: 8

```

Command

- Training
  - python3 bnn_pynq_train.py --network VGG_1W1A --experiments ./experiments
- Testing
  - python3 bnn_pynq_train.py --evaluate --network VGG_1W1A --resume ./ experiments /VGG_1W1A_xxxxxx/checkpoints/best.tar

©BOLEDU

Export Network

- Add the following two lines of code in train.py

```
if args.resume:
    print('Loading model checkpoint at: {}'.format(args.resume))
    package = torch.load(args.resume, map_location='cpu')
    model_state_dict = package['state_dict']
    model.load_state_dict(model_state_dict, strict=args.strict)
    vgg = model
    bo.export_finn_onnx(vgg, (1, 3, 32, 32),"./vgg_w1a1.onnx")
```

bnn_pynq_train.py

©BOLEDU

Part2 Requirements

- File
  - vgg_w1a1.onnx
- Screen Dump
  - Accuracy of this model on both Server and FPGA

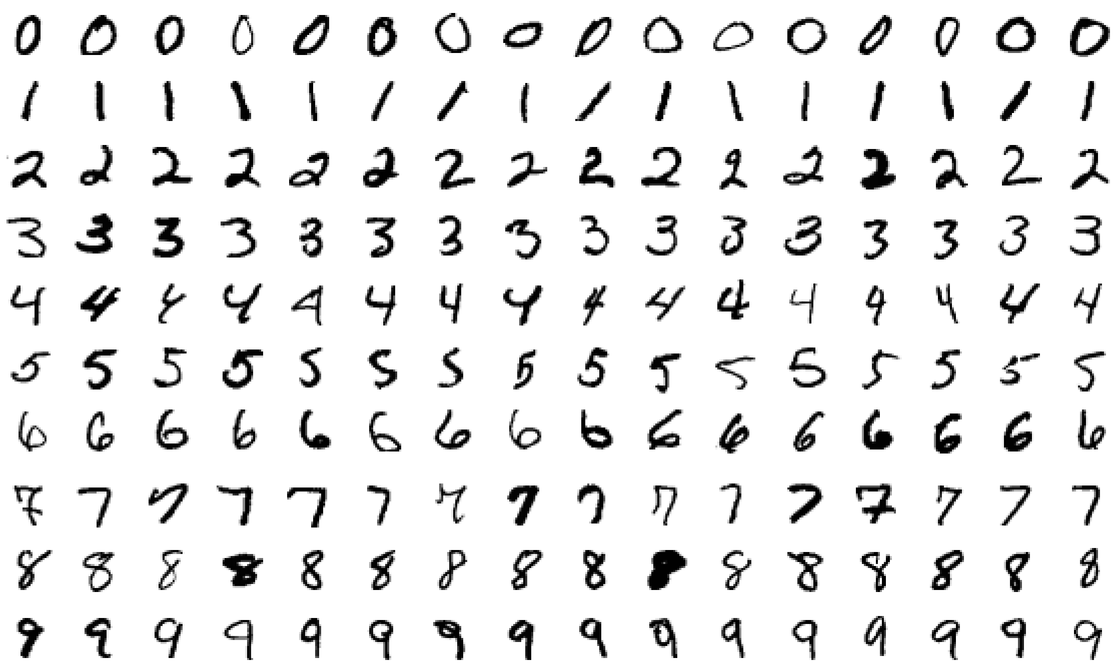
©BOLEDU

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance improvement
- Report & Submission

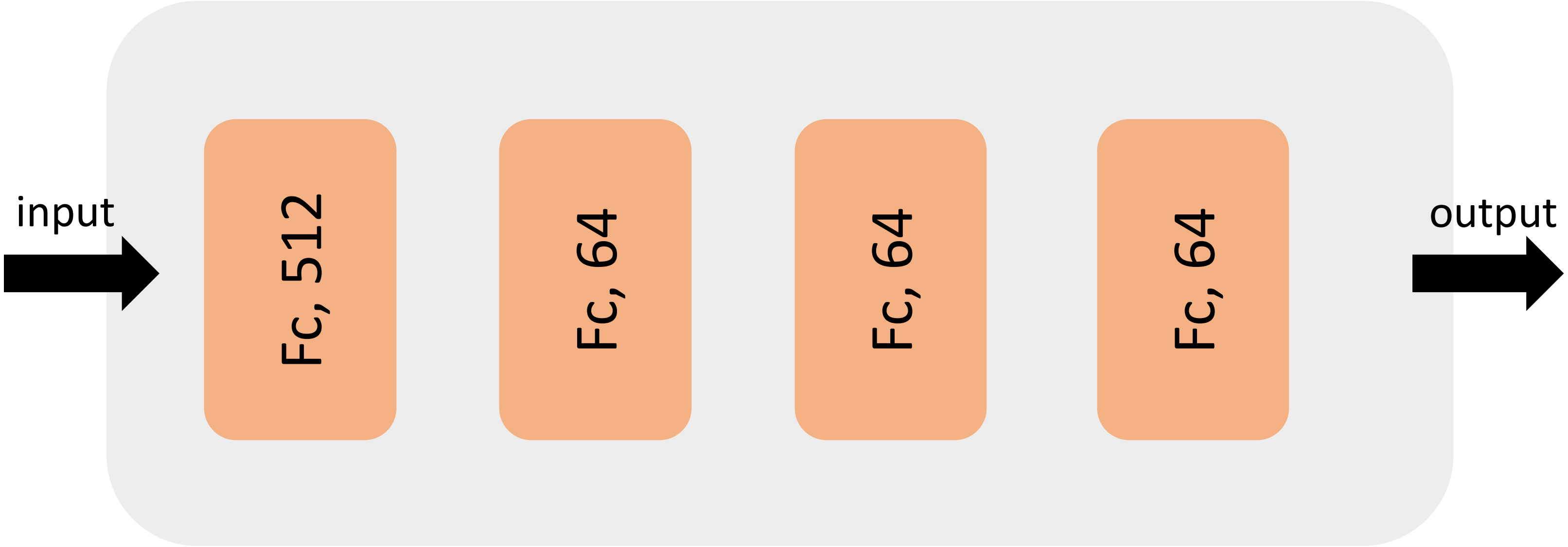
Handwritten Digit Classification

- Fully-connected network trained on the MNIST data set



©BOLEDU

Network Architecture



©BOLEDU

Folding: Adjusting the Parallelism

- We can set folding factors for each layer, controlled by the **PE** (parallelization over outputs) and **SIMD** (parallelization over inputs) parameters.

```
fc_layers = model.get_nodes_by_op_type("StreamingFCLayer_Batch")
# (PE, SIMD, in_fifo_depth, out_fifo_depth, ramstyle) for each layer
config = [
    (16, 49, 16, 64, "block"),
    (8, 8, 64, 64, "auto"),
    (8, 8, 64, 64, "auto"),
    (10, 8, 64, 10, "distributed"),
]
for fcl, (pe, simd, ififo, ofifo, ramstyle) in zip(fc_layers, config):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFODepth", ififo)
    fcl_inst.set_nodeattr("outFIFODepth", ofifo)
    fcl_inst.set_nodeattr("ram_style", ramstyle)

# set parallelism for input quantizer to be same as first layer's SIMD
inp_qnt_node = model.get_nodes_by_op_type("Thresholding_Batch")[0]
inp_qnt = getCustomOp(inp_qnt_node)
inp_qnt.set_nodeattr("PE", 49)
```

©BOLEDU

Throughput Test on PYNQ Board

- FINN provides the **throughput_test_remote** function for the throughput test.

```
from finn.core.throughput_test import throughput_test_remote

model = ModelWrapper(build_dir + "/tfc_w1_a1_pynq_deploy.onnx")
res = throughput_test_remote(model, 10000)
print("Network metrics:")
for key in res:
    print(str(key) + ": " + str(res[key]))
```

```
Network metrics:
runtime[ms]: 10.43391227722168
throughput[images/s]: 958413.2714850444
DRAM_in_bandwidth[Mb/s]: 751.3960048442748
DRAM_out_bandwidth[Mb/s]: 0.9584132714850445
fclk[mhz]: 100.0
N: 10000
```

©BOLEDU

Adjusting the Parallelism

- Try to set Layer2 PE and SIMD number to 1 (Exp1)
- Try your best to achieve the best performance by adjusting the PE and SIMD. (Exp2)
  - Try 3 sets at least.(List them in a table)

©BOLEDU

Part3 Requirements

- Screen Dump
  - Model Accuracy (At least 90%)
  - All of your performance results
- List all your results in a table
- Write down your observations and explain possible reasons for this result (Exp1)
- Explain your methodology for adjusting the folding factors and analyze the results. (Exp2)

©BOLEDU

Outline

- Part1: End-to-End FINN Flow for a Simple Convolutional Net
- Part2 : VGG on CIFAR-100
- Part3: Performance Improvement
- Questions
- Report & Submission

Question1

- Please draw the circuit diagrams for the following two PE and SIMD configuration.
  - PE: 4 SIMD: 1
  - PE: 8 SIMD: 8

©BOLEDU

Question2

- Following the previous question, suppose we need to calculate a 8 x 8 matrix multiplication; please calculate the latency for the two configurations.
- You have to draw some pictures to explain how the calculations are allocated to the hardware.

©BOLEDU

Question3

- For a $H \times W \times C_i$ feature map, and given C_0 stride 1 filters of size $K \times K$, what is the shape of the filter matrix and image matrix? (Assuming the input images are padded for simplicity, i.e. the output resolution is still H x W.)

©BOLEDU

Question4

- In this section, the image feature map in the figure is of size ***2 x 3***. However the real application the image might be of size ***224 x 224***. In this case, can we deal with the whole image? Are there any practical solutions in the perspective of hardware design?

©BOLEDU

Grading

- Part1: 20%
- Part2: 30%
- Part3: 30%
- Questions : 20%

©BOLEDU

Submission (1/2)

- **Hierarchy:**
  - GroupID_lab_FINN/
    - Part1/
      - All of the Screen Dumps
    - Part2/
      - All of the Screen Dumps
    - Part3/
      - All of the Screen Dumps
    - Question.pdf
    - Report.pdf

The file name of the screen dump should be comprehensible.

©BOLEDU

Submission (2/2)

- Compress all above files in a single zip file named GroupID_lab_FINN.zip
- Submit: Please ask your TAs where to submit.
- Due: 2022/5/9 (Mon.) 23:59

©BOLEDU

Bashrc File Configuration

- Only needs to be set for the first time log in.
- Open bashrc file
  - vi ~/.bashrc
- Create build directory in your home
  - mkdir ~/build
- Add the following codes at the end of bashrc file
  - export FINN_HOST_BUILD_DIR=/path/to/your/model/build
  - export FINN_XILINX_PATH =/tools/Xilinx
  - export FINN_XILINX_VERSION =2020.1
  - export PYNQ_BOARD=Pynq-Z2
  - PYNQ_IP=xxx.xxx.xxx.xxx
  - source /tools/Xilinx/Vivado/2020.1/settings64.sh

©BOLEDU