AAHLS Final Project:

High Accuracy Image Classifier Inference on PYNQ-Z2

Team: T8
B07901073 吳秉軒 R11943012 曾維雋 R11943043 潘奕亘

A Overview

In this report, we will describe how we implement the final project in AAHLS, including problem statement, training process, model selection, FINN transform flow.

B Problem statement

B.1 Introduction

- Dataset
  The CIFAR-10 dataset consists of 60000 32x32 color images in 10 classes, with 6000 images per class. There are 50000 training images and 10000 test images.
- Model
  We choose Resnet rather than VGG because Resnet has higher accuracy when having same parameter size, therefore suitable for edge device application.

B.2 Goal

Our goal is to implement two quantized Resnet models that has high accuracy in image classification task. The first version is a lightweight model that can implement on a smaller device with an acceptable accuracy, and the second version is a max-utilized model that fully utilized the hardware resource on PYNQ and can acquire higher accuracy.
Also, in the procedure of finding the model that satisfy our requirement, we want to find a methodology of tuning our model's parameter.

C Model training & exporting

C.1 Training setting

Besides the architecture of model, we use the same setting (listed below) for training during the whole project.

- Preprocessing:

1. RandomCrop
2. RandomHorizantalFlip
3. Normalize

- Optimizer: SGD
  1. Learning rate: 0.1
  2. Momentum: 0.9
  3. Weight decay: 0.0005
  4. Scheduler: CosineAnnealingLR
- Batch size: 128
- Epoch: 200

C.2 QNN Transformation

To produce quantized model for FPGA, we have to use QNN library to implement our model. There are three main differences between Pytorch model and QNN model:

- QuantIdentity layer:
  Since the input of model might not be quantized (e.g., float32, int32), we have to add an additional QuantIdentity layer at the beginning of model to transform input value. In addition, unlike normal CNN model (e.g., VGG), ResNet has shortcuts to produce more accurate results, and we also have to add QuantIdentity layer before adding two layers in shortcut.
- QNN layer:
  Luckily, QNN support most of the standard Pytorch layers, such as Conv2d, ReLu, and so on. Therefore, we can simply replace those layers with the corresponding layer in QNN library.
- Bit width setting:
  For the layers in QNN library, they have similar arguments in Pytorch, such as kernel, padding, stride, and so on. However, layers from QNN have additional bit_width argument to set quantized weight parameters or activation function.

Last, after training a QNN model, we can use FINNManager to generate onnx files for FPGA acceleration.

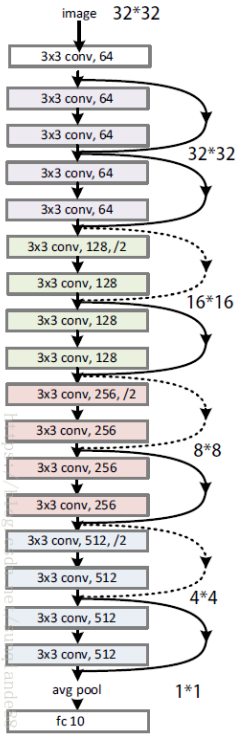
D Model selection

According to our goal, we want to implement two versions of quantized ResNet. For both versions, we start from ResNet18 with following spec:

- Model: ResNet18-4w4a

- Accuracy: 92.54%
- Total parameters: 11,164,352

Layer	Num of Param
Conv0	1,728
Group1	147,456
Group2	524,288
Group3	2,097,152
Group4	8,388,608
Fully-connected	5,120



D.1 Light-weight model

Our first objective is to implement a light-weight ResNet model with an acceptable accuracy (~85%).

To build a light-weight model, the most important thing is to identify non-critical resources that have least impact on accuracy. We modify our model based on following factors:

1. Layer:

   According to ResNet18-4w4a's spec, our first observation is that Group4 accounts for almost 75% of the number of parameters. Furthermore, there are two duplicands in each group, which also occupy half of the model.

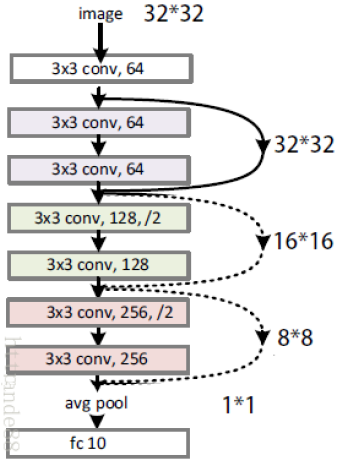
   Therefore, we first try to remove both parts of the ResNet18 and get another ResNet8 model. The results are listed below:

   Model: ResNet8-4w4a

   Accuracy: 89.98%

   Total parameter: 1,224,896

Layer	Num of Param
Conv0	1,728
Group1	73,728
Group2	229,376
Group3	917,504
Fully-connected	2,560



2. Channel & Bit width:

Next, we try to reduce channel and bit width. For resources, channel has squared relation with the number of parameters, while bit width only has linear relation. And for accuracy, we have observed that channel effects accuracy more than bit width does.

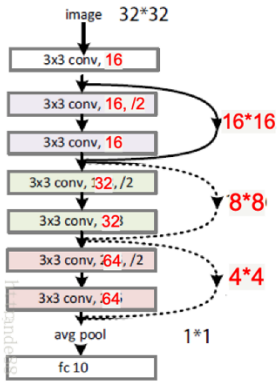
Finally, after try-and-error, our light-weight ResNet model's results are listed below (compared with ResNet18-4w4a):

Model: ResNet8-6w6a

Accuracy: 84.51%

Total parameter: 77,360

Layer	Num of Param	Decreased %
Conv0	432	75%
Group1	4,608	97%
Group2	14,336	97%
Group3	57,344	97%
Group4	0	100%
Fully-connected	640	87%



Although we haven't reached our baseline accuracy (~85%), we successfully reduce 99% of the original model size, which reaches our goal of implementing a light-weight model.

D.2 Max-utilized model

Our second objective is to implement a Resnet model that can approach best performance (~90%) by fully utilized the hardware resource on PYNQ. The main two procedure include:

- Truncate from Resnet18 to obtain a moderate size model that can fit in the PYNQ board by reduce layer or reduce channel.
- Slightly increase channel or bit width of some layer of the moderate size model to maximize the resource usage and approach ideal accuracy.

1. Moderate model finding

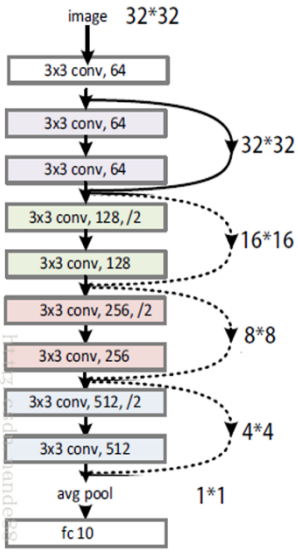
i. Resnet10-4w4a

We remove the duplicated block in Resnet18 so that the model contains 10 layers and the weight/activation bit width are 4 bits now, and the model size become 43.9% of Resnet18.

-Accuracy: 91.57%

-Total parameter: 4,897,472

Layer	Num of Param
Conv0	1,728
Group1	73,728
Group2	229,376
Group3	917,504
Group4	3,670,016
Fully-connected	5,120



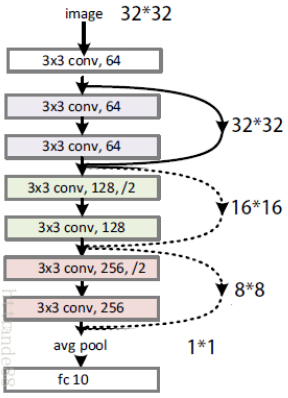
ii. Resnet8-4w4a

The model size of Resnet10 is still too large so we latter remove the group 4 of Resnet10 and obtain 8-layer model, and the model size become 25% of Resnet10 since group4 occupy 75% of the total model.

-Accuracy: 89.98%

-Total parameter: 1,224,896 (↓ 75%)

Layer	Num of Param
Conv0	1,728
Group1	73,728
Group2	229,376
Group3	917,504
Fully-connected	2,560

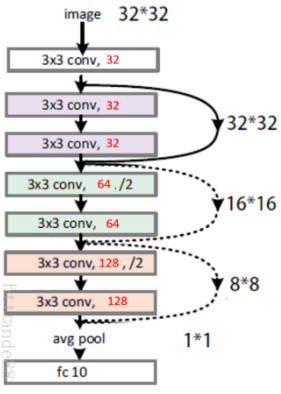


Latter we reduce half of the channel of all layers and the model size become 25% of original channel version, since the parameter size in a convolution layer is proportional to input channel and output channel, and now we get the moderate model Resnet8-4w4a that can implement on PYNQ.

-Accuracy: 87.63%

-Total parameter: 307,296 (↓ 75%)

Layer	Num of Param	Decreased %
Conv0	864	50%
Group1	18,432	75%
Group2	57,344	75%
Group3	229,376	75%
Fully-connected	1280	50%



2. Fully utilized hardware resource

i. Increase channel

We separately increase in channel of group1, group2 and group3 and observe the accuracy improvement, when increase the channel of group3, the accuracy improvement (0.87%) is most significant.

Best accuracy: 88.5% (↑0.87%)

Version	Conv0	Group1	Group2	Group3	FC	Acc %
Base	32	32	64	128	10	87.63
First group	48	48	64	128	10	88.09
Middle group	32	32	96	128	10	87.85
Last group	32	32	64	144	10	88.5

ii. Increase weight bit width

We test three version of increasing bit width, in the first version we increase the bit width of the first convolution and the last fully connected layer, in second version we increase the bit width of 3 groups of convolution layers, and last version we increase bit width of all layers.

We can observe that though the first version increase less parameter size than second version, it has better performance than second version, so the impact of changing bit width of the first and last layer is more significant, but after all increasing bit width of all layers can achieve the best performance with 1.7% accuracy improvement, so our max utilized model will be this Resnet8-5w5a.

Best accuracy: 89.33% (↑1.7%)

E Software issue

1. Onnx generation

i. Since there is residual block in Resnet model, so we need to add two layers after the shortcut path, we find out that we need to add QuantIdentity layer before adding these two layers otherwise we will encounter some error when generating onnx file.

ii. We train the quantized model on GPU to enhance the training speed, but when we need to generate onnx file, we need to move the trained model to CPU so that we can generate the onnx file successfully.

2. Onnx size

i. We observe that the onnx size will be proportional to the parameter size of the model, but the small model size doesn't represent that we can successfully synthesize on PYNQ. Except for model parameter size, it also involve model architecture and computation size, since the computation size is larger in the front layers, we can't use large channel in front layers otherwise the computation size will be too large and we fail to synthesize the model on PYNQ.

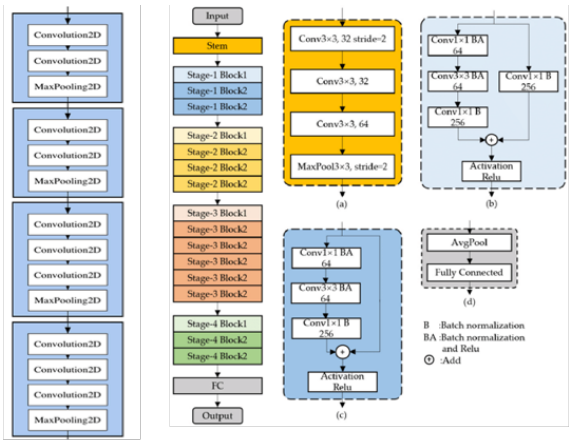
F FINN transform flow

F.1 Model comparison

- VGG architecture

  For VGG neural network, the VGG16 and VGG19 are common to use. In these usual architectures, they are composed of convolution, maxpooling and fully connected layers. These operations are also common in other neural network, including ResNet model.
- ResNet architecture
- ResNet models are also composed of convolution, maxpooling and fully connected layers. However, there are some operations apart from VGG architecture. These operations are convolution short-cut path, identity short-cut path and average pooling. Hence, the reason why we should build our own FINN flow is to tackle these additional operations which are not included in VGG architecture.

Layer	VGG	ResNet
CONV	V	V
Maxpool	V	V
FC	V	V
Avgpool		V
Short-cut		V



Architecture of VGG (left) vs. ResNet (right)

F.2 Flow

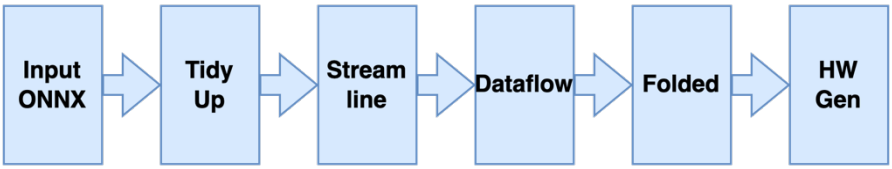
To use FINN, which is a tool to turn NN model to HLS code and even hardware IP, we mainly divide the transform flow into 5 stages. After import the onnx file, respectively do tidy-up, streamlining, dataflow, folding

and hardware generation to complete the model transformation.

Shortly introduce these procedures. First step is to tidy-up the model. The model would be transformed to initial topology graph. Then, we streamline the output from previous step. By this step, we convert the graph to matrix-matrix multiply operations, where one of the matrices is generated by sliding a window over the input image.

Next, we are going to fit the model on fpga board. Hence, dataflow is implemented. There are some important layer name we can know, such as MatrixVectorActivation following with ConvolutionInputGenerator being able to resolve convolution operations in original model. After dataflow, folding model is bound to implement because the resource on fpga is limit. We can set multiple parameters to fold model, such as PE, SIMD, FIFO depth and etc.

Finally, choose the hardware we want to deploy and do hardware generation step. In this project, we originally tried to use KV260 board to do inference; however, we didn't find the related library in FINN. Thus, we all implement on PYNQ-Z2.



F.3 Features

- Streamlined

  According to model comparison, in this step we should tackle with the shot-cut path operations in ResNet models. We can divide streamlining step into 2 parts. First part is to linear streamlining and second part is non-linear streamlining.

  For linear streamlining, the procedure is likely to VGG's procedure. However, non-linear streamlining is different to VGG's procedure. It can turn the short-cut path into available hardware. The main command is named MoveLinearPastEltwiseAdd().
- Dataflow

  Another feature in our project is that we implement quantized neural

network instead of binarized neural network. Hence, some commands about dataflow should be substituted. Such as InferBinearyMatrixVectorActivation() should be replaced with InferQuantizedMatrixVectorAcitvation().

- Folded
  Although the parameter of folding are configurable, there are some rules for them. We list some common constraints. First is that SIMD must divide IFMChannels, which means that the value of SIMD cannot be arbitrary. Second is that MatrixVectorActivation requires SIMD >= MW/1024, which means the value of SIMD should be higher than certain amount to accelerate the operations.

  Besides, we also found that FIFO depth doesn't influence the resource utilization much.

G FINN transform issue

G.1 Streamlined : Assertion error

As we do streamlining, there is an assertion error which is 'Signded output requires actval<0'. The solution is to add AbsorbSignBiasIntoMultiThreshold() command to the beginning of streamlining. By doing this, the tool would absorb signed integer into MultiThreshold and re-evaluate the output type.

G.2 Folded : Folded output shape is not consistent

In this step, we set multiple hardware related parameters to make model more likely fitting on the board. However, as we add InsertDWC() and InsertFIFO() this 2 commands in transform, the error about folded output shape being not consistent would be shown. DWC is the abbreviation of data width converter. Hence, we should eliminate these 2 commands. The FIFO depth would be set later in this procedure.

H FINN implementation results

In this section, we show three main implementation results to represent our overall project flow.
In addition, there are more attempts on our way to get final results, but they are too trivial to show on the report.

H.1 High accuracy version1: 90.11% (fail)

We start from the ResNet18. After some layers and channels were truncated, we got a smaller model whose accuracy is 90.11%. The architecture of model is shown below. The bit width of weight is 4 and the number of channels in first layer is 64. By this setting, the hardware resource is over-utilized. We can see LUT used 121% and BRAM used 127%

Layer	W bit	Shape	Channel
Conv0	4	32x32	3
Conv1_0	4	32x32	64
Conv1_1	4	16x16	64
Conv1_2	4	32x32	64
Conv2_0	4	16x16	64
Conv2_1	4	16x16	128
Conv2_2	4	8x8	64
Conv3_0	4	16x16	128
Conv3_1	4	4x4	256
Conv3_2	4	8x8	128

Element	ResNet10		PYNQ-Z2
LUT	64463	121%	53200
LUTRAM	6630	38%	17400
FF (slice)	96921	91%	106400
BRAM	178	127%	140
BUFG	1	3%	32

H.2 High accuracy version2 : 89.33% (pass)

Through the high accuracy version1, we know more about how to scale our neural network model. To fit our model into PYNQ-Z2, we first truncate the half of output channel of layer which is now 32. However, with view to retaining the model accuracy, we slightly increase the bit width of weight to 5 bits. Finally, our max-utilized version model can attain 89.33% accuracy. Related setting is shown below.

Layer	W bit	Shape	Channel
Conv0	5	32x32	3
Conv1_0	5	32x32	32
Conv1_1	5	32x32	32
Conv2_0	5	32x32	32
Conv2_1	5	16x16	64
Conv2_2	5	32x32	32
Conv3_0	5	16x16	64
Conv3_1	5	8x8	128
Conv3_2	5	16x16	64

Element	ResNet8		PYNQ-Z2
LUT	40593	76%	53200
LUTRAM	3150	18%	17400
FF (slice)	63355	60%	106400
BRAM	76	54%	140
BUFG	1	3%	32

H.3 Lightweight version : 84.51% (pass)

Furthermore, the other version we would achieve is a lightweight model with about 85% accuracy. Hence, like the method proposed above, we cut down the number of channel of first layer to 16. Also, slightly increase the bit width of weight to retain some accuracy. Our second version achieves

84.51% and costs less than half of hardware resource on PYNQ-Z2.

Layer	W bit	Shape	Channel
Conv0	6	32x32	3
Conv1_0	6	32x32	16
Conv1_1	6	16x16	16
Conv1_2	6	32x32	64
Conv2_0	6	16x16	16
Conv2_1	6	8x8	32
Conv2_2	6	16x16	16
Conv3_0	6	8x8	32
Conv3_1	6	4x4	64
Conv3_2	6	8x8	32

Element	ResNet8		PYNQ-Z2
LUT	35617	67%	53200
LUTRAM	2707	15%	17400
FF (slice)	46435	44%	106400
BRAM	29	21%	140
BUFG	1	3%	32

I Conclusion

To implement two versions of ResNet model, we try and analysis the relation between accuracy performance and parameter resources.

For ResNet, we found that the last convolution layer in the most critical layer for parameter resources. In addition, channel has more impact than bit width on both resources and accuracy performance.

To implement ResNet on PYNQ-Z2, we recommend setting the number of layers at 10. And 4~6 bit width for each layer will be enough for accuracy.

For hardware implementation, we use FINN to turn NN model into HDL. We build a specific flow for ResNet model owing to that there are some distinct features for ResNet.

After several times of hardware deployment, we found that the HW utilization at most costs 80%. If over this utilization, routing is more likely to fail.

Finally, we successfully fulfilled 2 desired designs, whose accuracy are respectively 84.51% and 89.33%.

Performance	Accuracy	Lightweight
LUT	76%	67%
LUTRAM	18%	15%
FF (slice)	60%	44%
BRAM	54%	21%
BUFG	3%	3%
Accuracy	89.33%	84.51%

J Github link

- https://github.com/PAN-YI-HSUAN/2022_HLS/tree/main/ResNet-Image_Classifier_on_PYNQZ2

K Reference

- https://github.com/Xilinx/brevitas
- https://xilinx.github.io/finn/
- https://finn.readthedocs.io/en/latest/source_code/finn.transformation.fpgadataflow.html