# AI Quantizer and AI Compiler – Caffe

Vitis AI Development Environment 2.0

## Abstract

This lab illustrates the Vitis™ AI quantizer and Vitis AI compiler flow for the Caffe framework. You will explore the required options for these tools and then run the tools.

This lab should take approximately 60 minutes.

## CloudShare Users Only

You are provided three attempts to access a lab, and the time allotted to complete each lab is 2X the time expected to complete the lab. Once the timer starts, you cannot pause the timer. Also, each lab attempt will reset the previous attempt—that is, your work from a previous attempt is not saved.

## Objectives

After completing this lab, you will be able to:

* Describe the AI quantizer flow and AI compiler flow
* Quantize a pre-trained Caffe model
* Compile the quantized model for the DPUCZDX8G architecture
* Verify the files generated after quantization and compilation for the Caffe model network

## Introduction

There are two stages for developing deep learning applications: training and inference. The training stage is used to design a neural network for a specific task (such as image classification) using a huge amount of training data. The inference stage involves the deployment of a previously designed neural network to handle new input data not seen during the training stage.

The Vitis AI toolchain provides an innovative workflow to deploy deep learning inference applications on the DPU with the following four steps:

* Quantize the neural network model
* Compile the neural network model
* Program with the Vitis AI programming interface
* Run and evaluate the deployed DPU application

In this lab, you will be focusing on quantization and compilation.

Model Quantization

The Vitis AI quantizer supports the Caffe, TensorFlow (both 1.x and 2.x), and PyTorch frameworks. vai\_q\_caffe, vai\_q\_tensorflow, and vai\_q\_pytorch are the Vitis AI quantizer, where q stands for quantizer and caffe, tensorflow, or pytorch are the framework names.

Post training quantization (PTQ) requires only a small set of unlabeled images to analyze the distribution of activations. The running time of quantize calibration varies from a few seconds to several minutes, depending on the size of the neural network.

Generally, there is some drop in accuracy after quantization. However, for some networks (such as Mobilenet), the accuracy loss might be large. In this situation, quantization aware training (QAT) can be used to further improve the accuracy of the quantized models.

QAT requires the original training dataset. Several epochs of finetuning are needed and the finetune time varies from several minutes to several hours. It is recommended to use small learning rates when performing QAT.

For PTQ, the cross-layer equalization algorithm is implemented. Cross-layer equalization can improve calibration performance, especially for networks including depth-wise convolution.

Note: For the Vitis AI development environment 1.4 onwards, the term "quantize calibration" is replaced with "post training quantization", and "quantize finetuning" is replaced with "quantization aware training."

| Frameworks | Versions | Post Training Quantization (PTQ) | Quantize Aware Training (QAT) | Fast Finetuning (Advanced Calibration) |
| --- | --- | --- | --- | --- |
| TensorFlow 1.x | Based on 1.15 | Yes | Yes | No |
| TensorFlow 2.x | Supports 2.3 | Yes | Yes | Yes |
| PyTorch | Supports 1.2 – 1.9 | Yes | Yes | Yes |
| Caffe | – | Yes | Yes | No |

Compiling

The Vitis AI compiler is the unified interface to a compiler family targeting the optimization of neural network computations to a family of DPUs.

Each compiler maps a network model into a highly optimized DPU instruction sequence.

Understanding the Lab Environment

Customizable environment variables enable you to tailor your environment for specific machine configurations. The only environment variable (shown below) used in the customer training environment (CustEd\_VM) points to the training directory where all the lab files are located.

This environment variable can be customized according to your specific location and can be set for Linux systems in the /etc/profile file.

The following is the environment variable used in the customer training VM:

| Environment Variable Name | Description |
| --- | --- |
| $TRAINING\_PATH | Points to the space allocated for students to work through their labs. This directory includes prebuilt images and starting points for the labs and demos. In the customer training VM, $TRAINING\_PATH sets to the /home/xilinx/training directory. |

Note: Environment variables are not supported from the Vitis IDE GUI. When using this tool, you must manually replace $TRAINING\_PATH with the value of the variable, which in the customer training virtual machine, is /home/xilinx/training.

## General Flow

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Step 1:  Reviewing AI Quantizer Arguments |  | Step 2:  Running  the AI Quantizer |  | Step 3:  Reviewing AI Compiler Arguments |  | Step 4:  Running  the AI  Compiler |

Reviewing the AI Quantizer Arguments Step

The overall model quantization flow is outlined in the following figure.

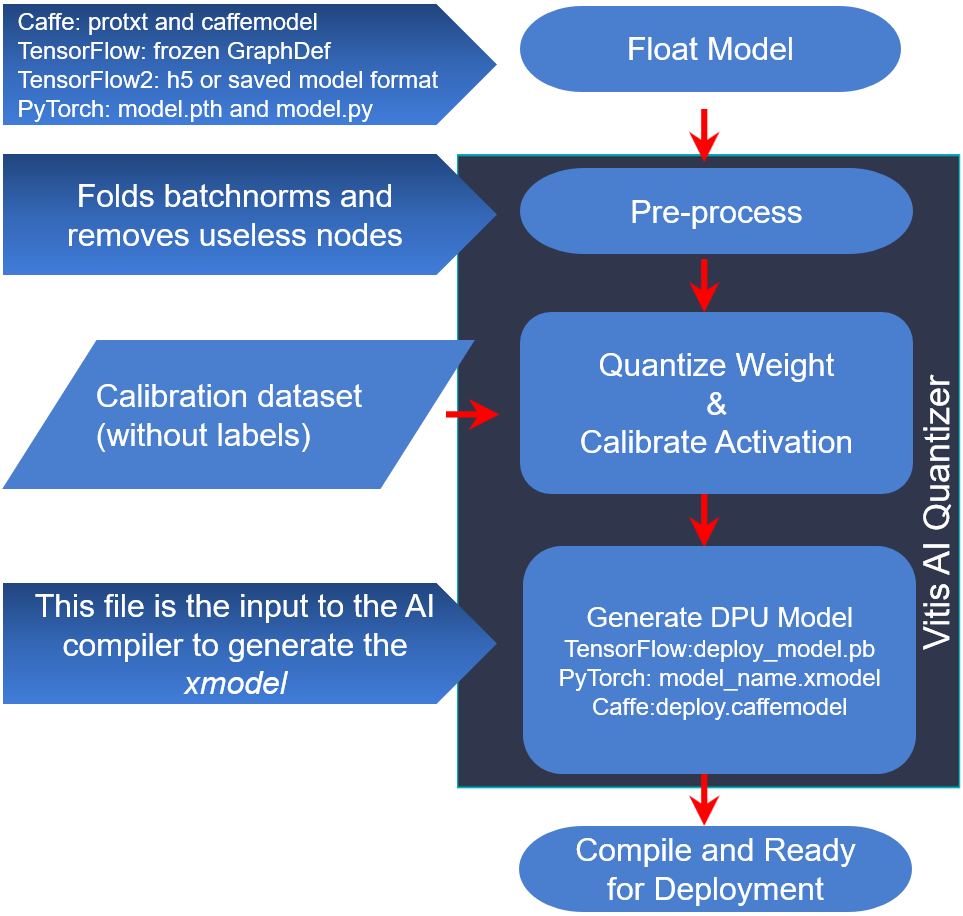


Figure 1‑: Vitis AI Quantizer Flow

The Vitis AI quantizer takes a floating-point model as input, performs pre-processing (folding batch norms and removing nodes not required for inference), and then quantizes the weights/biases and activations to the given bit width.

To capture activation statistics and improve the accuracy of quantized models, the Vitis AI quantizer must run several iterations of inference to calibrate the activations. A calibration image dataset input is therefore required. Generally, the quantizer works well with 100–1000 calibration images. This is because there is no need for back propagation; the un-labeled dataset is sufficient.

After calibration, the quantized model is transformed into a DPU-deployable model (named deploy\_model.pb for vai\_q\_tensorflow, model\_name.xmodel for vai\_q\_pytorch, or deploy.prototxt/deploy.caffemodel for vai\_q\_caffe), which follows the data format of a DPU.

This model can then be compiled by the Vitis AI compiler and deployed to the DPU. The quantized model cannot be taken in by the standard version of the Caffe, TensorFlow, or PyTorch framework.

1-1. Copy the lab files to the docker location.

1-1-1. Press <Ctrl + Alt + T> to open a new terminal window.

1-1-2. Enter the following command to copy the files to the specified directory:

[host]$ cp -rf $TRAINING\_PATH/vai\_q\_c /home/xilinx/Vitis-AI

Note: The Vitis AI tool is installed in the /home/xilinx/Vitis-AI directory in the Customer Training VM. If the tool is installed in a different location in your environment, use that install path.

The lab folder under vai\_q\_c contains the following:

* cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0
* This contains the pre-trained model.
* images directory
* This contains the training dataset.
* 1\_caffe\_quantize\_for\_edge.sh
* This script calls the vitis-ai quantizer for Caffe with all the required arguments.
* 2\_caffe\_compile\_for\_edge.sh
* This script calls the vitis-ai compiler for Caffe with all the required arguments.
* caffe\_calib.txt
* For quantize calibration, calibration data without a label is enough. But due to the implementation, a image list file with two columns is required. Just set the second column to a random value or zero. In this case, it is set to one.

1-2. Load the CPU image from the Docker hub.

1-2-1. Press <Ctrl + Alt + T> to open a new terminal window.

1-2-2. Enter the following commands to load the CPU image from the Docker hub:

[host]$ cd Vitis-AI

[host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:latest

If there is a newer version of the Vitis AI tools that has been released, entering the above command will download the latest version (that is, not VAI 2.0). In order to use the Vitis AI 2.0 environment, enter the command with the docker image tag. You can find the tag from https://hub.docker.com/r/xilinx/vitis-ai-cpu/tags?page=1&ordering=last\_updated.

For example: [host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:<Docker\_tag>

To use 2.0: [host]$ ./docker\_run.sh xilinx/vitis-ai-cpu:2.0

Note: Keep clicking to accept the terms and agreements and then enter 'y'.

The terminal output should be similar to the following:

==========================================

\_\_ \_\_\_ \_ \_ \_\_\_\_\_

\ \ / (\_) | (\_) /\ |\_ \_|

\ \ / / \_| |\_ \_ \_\_\_ \_\_\_\_\_\_ / \ | |

\ \/ / | | \_\_| / \_\_|\_\_\_\_\_\_/ /\ \ | |

\ / | | |\_| \\_\_ \ / \_\_\_\_ \ \_| |\_

\/ |\_|\\_\_|\_|\_\_\_/ /\_/ \\_\\_\_\_\_\_|

==========================================

Docker Image Version: 2.0.0.1103 (CPU)

Vitis AI Git Hash: 06d7cbb

Build Date: 2022-01-12

For TensorFlow 1.15 Workflows do:

conda activate vitis-ai-tensorflow

For Caffe Workflows do:

conda activate vitis-ai-caffe

For PyTorch Workflows do:

conda activate vitis-ai-pytorch

For TensorFlow 2.6 Workflows do:

conda activate vitis-ai-tensorflow2

Vitis-AI /workspace >

1-3. Activate the Caffe conda environment.

1-3-1. Enter the following command to activate the Caffe conda environment:

conda activate vitis-ai-caffe

The terminal output should be similar to the following:

(vitis-ai-caffe) Vitis-AI /workspace >

1-4. Review the 1\_caffe\_quantize\_for\_edge.sh script.

1-4-1. Press <Ctrl + Alt + T> to open a new terminal window.

1-4-2. Enter the following command to review the quantizer arguments in the quantizer script:

[host]$ gedit /home/xilinx/Vitis-AI/vai\_q\_c/lab/  
1\_caffe\_quantize\_for\_edge.sh

The detailed commands used in the script are as shown below.

#For DPUCZDX8G

vai\_q\_caffe quantize -model ${CF\_NETWORK\_PATH}/float/trainval.prototxt \

-weights ${CF\_NETWORK\_PATH}/float/trainval.caffemodel \

-output\_dir ${CF\_NETWORK\_PATH}/vai\_q\_output\_dpu\_edge \

-calib\_iter 10 \

-test\_iter 10 \

-auto\_test \

-keep\_fixed\_neuron

Note: vai\_q\_caffe is the Vitis AI quantizer, where q stands for quantizer and caffe is the framework name.

Ensure that you specify the -keep\_fixed\_neuron option for vai\_q\_caffe because it is essential for the XIR-based compiler.

| Quantizer Argument | Description |
| --- | --- |
| model | Caffe floating-point network model prototxt file. |
| weights | Pre-trained Caffe floating-point network model caffemodel file. |
| output\_dir | Output directory location to store the generated output. |
| calib\_iter | Subset of the training set containing 100 to 1000 images. |
| test\_iter | Provides the number of test iterations. |

In this lab, you will be using the cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0 pre-trained model from the Vitis AI Model Zoo. The pre-trained model has been already downloaded and is located in the /home/xilinx/Vitis-AI/vai\_q\_c/lab directory.

You will find the pre-trained floating model under the directory cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0 > float.

| Application | Image Classification |
| --- | --- |
| Model | resnet50 |
| Name | cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0 |
| Framework | caffe |
| Backbone | resnet50 |
| Input Size | 224\*224 |
| OPS per image | 7.7G |
| Training Set | ImageNet training |
| Val Set | ImageNet validation |
| Float (Top1, Top5) | 0.7444/0.9185 |
| Quantized (Top1, Top5) | 0.7334/0.9131 |

With regards to the naming of cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0:

* cf specifies the Caffe framework.
* imagenet specifies the dataset.
* 224\_224 specifies the height of the input data x the width of the input data.
* 7.7G specifies the computation of the model; how many giga-operations per second (GOPs) per image.
* 2.0 specifies the Vitis AI development environment version.

In order for calibration and evaluation in the quantization phase to be conducted, images and labels from ImageNet have been downloaded and placed under the vai\_q\_c/lab/images directory.

ILSVRC2012\_val\_00000001.JPEG to ILSVRC2012\_val\_00000500.JPEG are used in this example.

1-5. Make sure that the the calibration and test iteration values are set to 2.

These values are set to reduce the quantization execution time.

1-5-1. Change the calibration and test iteration values in the 1\_caffe\_quantize\_for\_edge.sh script as shown below (in bold) if required:

#For DPUCZDX8G

vai\_q\_caffe quantize -model ${CF\_NETWORK\_PATH}/float/trainval.prototxt \

-weights ${CF\_NETWORK\_PATH}/float/trainval.caffemodel \

-output\_dir ${CF\_NETWORK\_PATH}/vai\_q\_output\_dpu\_edge \

-calib\_iter 2 \

-test\_iter 2 \

-auto\_test \

-keep\_fixed\_neuron

1-5-2. Save and close the 1\_caffe\_quantize\_for\_edge.sh file.

1-6. Set the calibration dataset and the image location required to generate the quantized model.

1-6-1. Enter the following command to set the calibration dataset and the image location in the pre-trained floating model trainval.prototxt file:

[host]$ gedit /home/xilinx/Vitis-AI/vai\_q\_c/lab/  
cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/float/trainval.prototxt

1-6-2. Update the following lines (line no. 18, 19 and 39, 40) as shown below:

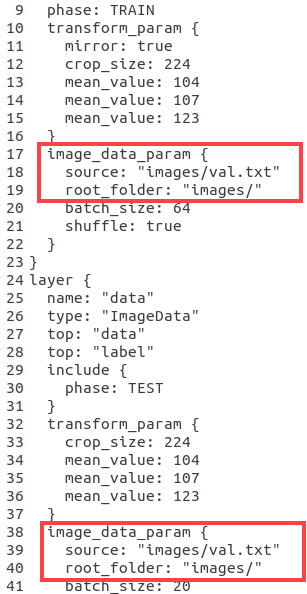


Figure 1‑: Calibration Dataset and Image Location

1-6-3. Save and close the file.

1-6-4. Close the terminal.

Running the AI Quantizer Step

2-1. Run the AI Quantizer tool by executing the 1\_caffe\_quantize\_for\_edge.sh script.

The script sets the environment variable CF\_NETWORK\_PATH and runs the AI Quantizer tool.

2-1-1. Go to the terminal where Caffe has been activated.

2-1-2. Enter the following commands to run the Vitis AI quantizer for Caffe:

(vitis-ai-caffe) Vitis-AI /workspace > cd vai\_q\_c/lab

(vitis-ai-caffe) Vitis-AI /workspace/vai\_q\_c/lab > sh 1\_caffe\_quantize\_for\_edge.sh

The quantizer tool now starts running. The execution time usually depends on the calibration dataset.

In this case, it should take approximately 23-25 minutes if you are using the VM.

If you are using CloudShare, it should take approximately 7-9 minutes.

After the quantization is over, you should see messages as shown below.

...

I0310 03:47:34.527132 100 net\_test.cpp:394] Test iter: 1/2, top-1 = 0.55

I0310 03:47:34.527185 100 net\_test.cpp:394] Test iter: 1/2, top-5 = 0.8

I0310 03:48:12.894712 100 net\_test.cpp:394] Test iter: 2/2, top-1 = 0.65

I0310 03:48:12.894765 100 net\_test.cpp:394] Test iter: 2/2, top-5 = 0.85

I0310 03:48:12.894778 100 net\_test.cpp:405] Test Results:

I0310 03:48:12.894786 100 net\_test.cpp:406] Loss: 0

I0310 03:48:12.894798 100 net\_test.cpp:421] top-1 = 0.6

I0310 03:48:12.894814 100 net\_test.cpp:421] top-5 = 0.825

I0310 03:48:12.894824 100 net\_test.cpp:450] Test Done!

I0310 03:48:12.952584 100 vai\_q.cpp:360] Start Deploy

I0310 03:48:27.895336 100 vai\_q.cpp:368] Deploy Done!

--------------------------------------------------

Output Quantized Train&Test Model: "cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/quantize\_train\_test.prototxt"

Output Quantized Train&Test Weights: "cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/quantize\_train\_test.caffemodel"

Output Deploy Weights: "cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.caffemodel"

Output Deploy Model: "cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.prototxt"

(vitis-ai-caffe) Vitis-AI /workspace/vai\_q\_c/lab >

After successful execution of the quantizer command, four files are generated in the output directory (vai\_q\_output\_dpu\_edge).

* The deploy.prototxt and deploy.caffemodel files are used as input files to the compiler.
* The quantize\_train\_test.prototxt and quantize\_train\_test.caffemodel files are used to test the accuracy on the GPU/CPU and can be used as input files to quantize finetuning.

Note: The quantize\_train\_test model can be used for INT8 evaluation, finetuning, or DPU result emulation. The deploy model can be used to generate DPU ELF file.

The vai\_q\_caffe output files are shown below.

| Serial Number | Name | Description |
| --- | --- | --- |
| 1. | deploy.prototxt | Quantized network description file for the Vitis AI compiler. |
| 2. | deploy.caffemodel | Quantized Caffe model parameter file (non-standard Caffe format) for the Vitis AI compiler. |
| 3. | quantize\_train\_test.prototxt | Quantized network description file for testing and finetuning. |
| 4. | quantize\_train\_test.caffemodel | Quantized Caffe model parameter file (non-standard Caffe format) for testing and finetuning. |

Reviewing the AI Compiler Arguments Step

The Vitis AI compiler (VAI\_C) is the unified interface to a compiler family targeting the optimization of neural network computations to a family of DPUs. Each compiler maps a network model into a highly optimized DPU instruction sequence.

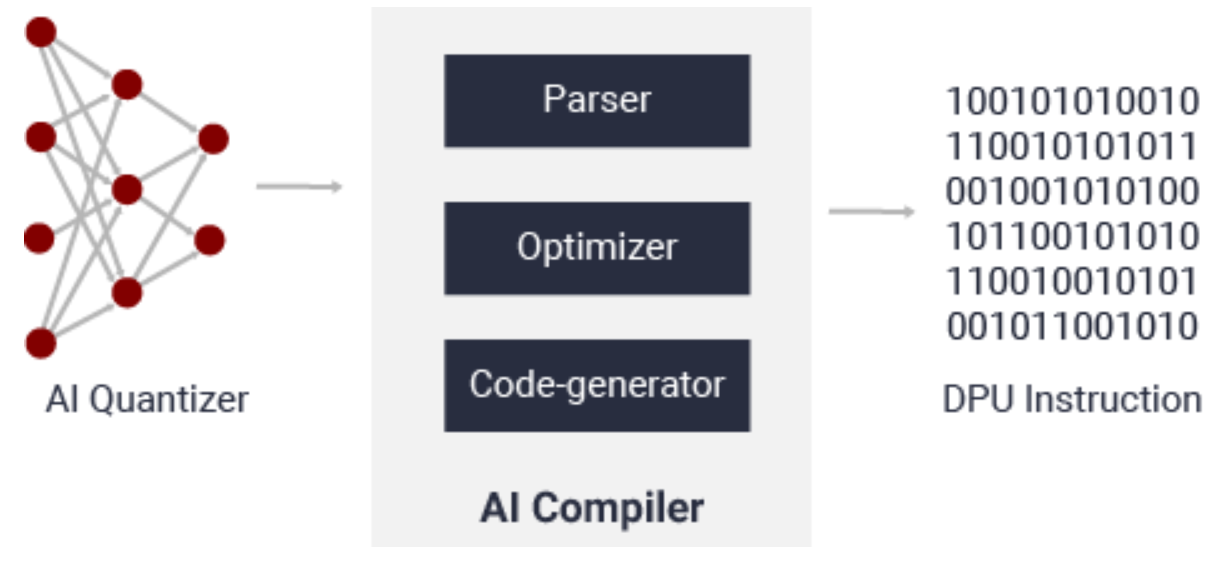


Figure 1‑: Vitis AI Compiler Framework

After parsing the topology of the optimized and quantized input model, the Vitis AI compiler constructs an internal computation graph as an intermediate representation (IR), which shows a corresponding control flow and data flow representation.

The compiler then performs multiple optimizations; for example, computation node fusion (such as when batch norm is fused into a preceding convolution), efficient instruction scheduling by exploiting inherent parallelism, or exploiting data reuse.

The Vitis AI Compiler generates the compiled model based on the DPU microarchitecture. There are a number of different DPUs supported in the Vitis AI environment for different platforms and applications. It is important to understand the relations between the available compilers and associated DPUs.

| DPU Name | Hardware Platform |
| --- | --- |
| DPUCZDX8G | Zynq® UltraScale+™ MPSoC |
| DPUCAHX8H | Alveo™ U50LV, U55C Data Center accelerator cards |
| DPUCADF8H | Alveo U200, U250 Data Center accelerator cards |
| DPUCVDX8G | Versal® ACAP VCK190 (Versal AI Core Series) evaluation kit |
| DPUCVDX8H | Versal ACAP VCK5000 evaluation kit |

Xilinx Intermediate Representation (XIR) is a graph-based intermediate representation of the AI algorithms which is designed for compilation and efficient deployment of the DPU on the FPGA platform. If you are an advanced user, you can apply whole application acceleration to allow the FPGA to be used to its maximum potential by extending the XIR to support customized IPs in the Vitis AI flow. It is the current foundation for the Vitis AI quantizer, compiler, runtime, and other tools.

The XIR-based compiler takes the quantized model (TensorFlow or Caffe) as the input. First, it transforms the input models into the XIR format as the foundation of the processes that follow. Most of the variations among different frameworks are eliminated and transferred to a unified representation in XIR.

The compiler then applies various optimizations to the graph and breaks up the graph into several subgraphs on the basis of whether the operation can be executed on the DPU. More architecture-aware optimizations are applied for each subgraph as required. For the DPU subgraph, the compiler generates the instruction stream and attaches to it.

Finally, the optimized graph with the necessary information and instructions for VART is serialized into a compiled xmodel file.

For PyTorch, the quantizer NNDCT outputs the quantized model in the XIR format directly. Use vai\_c\_xir to compile it.

For Caffe, vai\_q\_caffe generates a PROTOTXT (deploy.prototxt) and a MODEL (deploy.caffemodel). Ensure that you specify the -keep\_fixed\_neuron option for vai\_q\_caffe, which is essential for the XIR-based compiler.

3-1. Review the 2\_caffe\_compile\_for\_edge.sh script.

3-1-1. Press <Ctrl + Alt + T> to open a new terminal window.

3-1-2. Enter the following command to review the compiler arguments in the compiler script:

[host]$ gedit /home/xilinx/Vitis-AI/vai\_q\_c/lab/  
2\_caffe\_compile\_for\_edge.sh

The detailed commands used in the script are as shown below.

#!/bin/sh

NET\_NAME=resnet50

EDGE\_TARGET=ZCU102

DEPLOY\_MODEL\_PATH=vai\_q\_output\_dpu\_edge

EDGE\_ARCH=/opt/vitis\_ai/compiler/arch/DPUCZDX8G/${EDGE\_TARGET}/${EDGE\_TARGET}.json

export CF\_NETWORK\_PATH='cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0'

#DPUCZDX8G

vai\_c\_caffe --prototxt ${CF\_NETWORK\_PATH}/${DEPLOY\_MODEL\_PATH}/deploy.prototxt \

--caffemodel ${CF\_NETWORK\_PATH}/${DEPLOY\_MODEL\_PATH}/deploy.caffemodel \

--arch ${EDGE\_ARCH} \

--output\_dir ${CF\_NETWORK\_PATH}/vai\_c\_output\_${EDGE\_TARGET}/ \

--net\_name ${NET\_NAME} \

--options "{'save\_kernel':''}"

Common options for the Vitis AI compiler for the cloud and edge DPU are shown in the table below.

| Compiler Options | Description |
| --- | --- |
| --prototxt | Path of the Caffe prototxt file for the compiler vai\_c\_caffe. This option is only required when compiling the quantized Caffe model generated by vai\_q\_caffe. |
| --caffemodel | Path of the Caffe caffemodel file for the compiler vai\_c\_caffe. This option is only required when compiling the quantized Caffe model generated by vai\_q\_caffe. |
| --arch | DPU architecture configuration file for the Vitis AI compiler in JSON format. It contains the dedicated options for cloud and edge DPU during compilation. |
| --output\_dir | Path of the output directory of vai\_c\_caffe and vai\_c\_tensorflow after the compilation process. |
| --net\_name | Name of the DPU kernel for the network model after compilation by the Vitis AI compiler. |
| --options | The list for the extra options for the cloud or edge DPU in the format of 'key':'value'.  If there are multiple options to be specified, they are separated by ‘,’, and if the extra option has no value, an empty string must be provided.  For example:  --options "{'cpu\_arch':'arm32', 'dcf':'/home/edge-dpu/zynq7020.dcf', 'save\_kernel':''}" |

3-1-3. After completing your review, close the 2\_caffe\_compile\_for\_edge.sh file.

Running the AI Compiler Step

4-1. Run the AI compiler tool by executing the 2\_caffe\_compile\_for\_edge.sh script.

4-1-1. Go to the terminal where the quantizer had been executed.

4-1-2. Enter the following command to run the Vitis AI compiler for Caffe:

(vitis-ai-caffe) Vitis-AI /workspace/vai\_q\_c/lab > sh 2\_caffe\_compile\_for\_edge.sh

When a network model is compiled, the required options should be specified to the Vitis AI compiler. Once compilation is successful, the Vitis AI compiler will generate ELF object files and kernel information for deployment. These files are located under the folder specified by output\_dir.

You should see the following output after compilation has completed.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\* VITIS\_AI Compilation - Xilinx Inc.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

[INFO] Namespace(batchsize=1, inputs\_shape=None, layout='NCHW', model\_files=['cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.caffemodel'], model\_type='caffe', named\_inputs\_shape=None, out\_filename='/tmp/resnet50\_org.xmodel', proto='cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.prototxt')

[INFO] caffe model: /workspace/vai\_q\_c/lab/cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.caffemodel

[INFO] caffe model: /workspace/vai\_q\_c/lab/cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_q\_output\_dpu\_edge/deploy.prototxt

[INFO] parse raw model :100%|█| 194/194 [00:24<00:00, 7.77it/s]

[INFO] infer shape (NCHW) :100%|█| 194/194 [00:00<00:00, 450.86it/s]

[INFO] infer shape (NHWC) :100%|█| 194/194 [00:00<00:00, 1358.38it/s]

[INFO] perform level-1 opt :100%|█| 3/3 [00:00<00:00, 69.30it/s]

[INFO] infer shape (NHWC) :100%|█| 196/196 [00:00<00:00, 4759.96it/s]

[INFO] generate xmodel :100%|█| 196/196 [00:00<00:00, 486.42it/s]

[INFO] dump xmodel: /tmp/resnet50\_org.xmodel

[UNILOG][INFO] Compile mode: dpu

[UNILOG][INFO] Debug mode: function

[UNILOG][INFO] Target architecture: DPUCZDX8G\_ISA0\_B4096\_MAX\_BG2

[UNILOG][INFO] Graph name: deploy, with op num: 412

[UNILOG][INFO] Begin to compile...

[UNILOG][INFO] Total device subgraph number 3, DPU subgraph number 1

[UNILOG][INFO] Compile done.

[UNILOG][INFO] The meta json is saved to "/workspace/vai\_q\_c/lab/cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_c\_output\_ZCU102/meta.json"

[UNILOG][INFO] The compiled xmodel is saved to "/workspace/vai\_q\_c/lab/cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_c\_output\_ZCU102//resnet50.xmodel"

[UNILOG][INFO] The compiled xmodel's md5sum is 6b9445d488485f3565f365948dd43945, and has been saved to "/workspace/vai\_q\_c/lab/cf\_resnet50\_imagenet\_224\_224\_7.7G\_2.0/vai\_c\_output\_ZCU102/md5sum.txt"

(vitis-ai-caffe) Vitis-AI /workspace/vai\_q\_c/lab >

The compiler creates two files in the OUTPUTPATH directory (vai\_c\_output\_ZCU102):

* Compiled xmodel: resnet50.xmodel
* For run time: meta.json

## Summary

In this lab, you reviewed the inputs for the Vitis AI quantizer for Caffe and then ran the Vitis AI compiler for Caffe by using the output from the quantizer.