Part 4a1. DeePhi ML design flow

The flow: Input and Output files

- DECENT tool needs floating point NN network model, Pre-training weights and calibration dataset of your DNN in Caffe format to work on
  - **float.prototxt**: Input floating point model
  - **float.caffemodel**: Pre-training weights in floating point
  - **calibration** dataset: calibration dataset is usually a subset of training set containing about 100~1000 pictures
- Once the quantization is done, it will generated two model files named **deploy.prototxt** and **deploy.caffemodel**, these are input files for DNNC compiler:
  - deploy.prototxt: fixed point quantized network description file
  - deploy.caffemodel: fixed point quantized weights Caffe model file (note that this is not a standard Caffe format)

The Quantization flow in 5 steps

- 1. Compress neural network model (from host side) with **decent**
  - command example for miniVggNet case: `source decent_miniVggNet.sh`
- 2. Compile neural network model (from host side) with **dnnc**
  - command example for miniVggNet case: `source dnnc_miniVggNet.sh`
- 3. Edit the **main.cc** program application (from host side)
- 4. Compile hybrid application (from target side) with **make** utility
  - command examples for miniVggNet case: `make clean; make build`
- 5. Run hybrid application (from target side)
  - command example for miniVggNet case: `./miniVggNet`

Some terminology for step2

- **Kernel id**: id of current kernel. Every kernel has a unique id assigned by DNNC.
- **Kernel name**: name of current kernel. Each kernel supported by DPU will have a corresponding ELF object file with name that is same with kernel name prefixed by “dpu_” with extension ”.elf”. The kernel name will be used in your application code, thus N2Cube can identify different kernels correctly.
- **Type**: the kernel type. Three types of kernel are supported by DNNC, see Table 6 for detail.
- **Nodes**: All nodes included in current kernel. For kernel supported by DPU, we use “NA” instead to avoid printing all names.
- **Input nodes**: All input nodes of current kernel. For kernel not supported by DPU, programmer must get output of preceding kernel through output nodes and feed them into input nodes of current nodes using APIs provided by N2Cube.
- **Output nodes**: All output nodes of current kernel. The output nodes’ address and size can be extracted using APIs provided by N2Cube.

Quantization of miniVggNet

miniVggNet project files

For the calibration images data use a soft link (to save HD space):

- `cd <home>/ML/cifar10/deephi/miniVggNet`
- `mkdir quantiz`
- `mkdir quantiz/data`
- `ln -s /home/danieleb/ML/cifar10/input/cifar10_jpg/calib ./quantiz/data/calib`

- `cd zcu102`
- `ln -s test_images ./baseline/test_images`
- `ln -s test_images ./pruned/test_images`

```
(caffe_python_2) danieleb@CentOS63-x86-64:~$ cd ML/cifar10/deephi/
(caffe_python_2) danieleb@CentOS63-x86-64:~/ML/cifar10/deephi$ tree -d
.
└── miniVggNet
    ├── pruning
    │   ├── quantiz
    │   │   ├── data
    │   │   │   └── calib -> /home/danieleb/ML/cifar10/input/cifar10_jpg/calib
    │   │   ├── decent_output
    │   │   ├── dnnc_output
    │   │   └── rpt
    │   ├── regular_rate_0
    │   │   └── snapshots
    │   ├── regular_rate_0.3
    │   │   └── snapshots
    │   ├── regular_rate_0.7
    │   │   └── snapshots
    │   └── rpt
    ├── quantiz
    │   ├── data
    │   │   └── calib -> /home/danieleb/ML/cifar10/input/cifar10_jpg/calib
    │   ├── decent_output
    │   ├── dnnc_output
    │   └── rpt
    └── zcu102
        ├── baseline
        │   ├── model
        │   │   └── arm64_4096
        │   ├── rpt
        │   ├── src
        │   └── test_images -> ../test_images
        ├── pruned
        │   ├── model
        │   │   └── arm64_4096
        │   ├── rpt
        │   ├── src
        │   └── test_images -> ../test_images
        └── test_images

35 directories
(caffe_python_2) danieleb@CentOS63-x86-64:~/ML/cifar10/deephi$
```

miniVggNet project files

- There are three main folders:
- **`quantiz`:** it contains the scripts and related resuts to quantize the CNN (which is baseline, that is, not pruned). Logfiles are stored in **`rpt`**
- **`pruning`:** it contains the scripts and related files to prune the CNN first, and then to quantize it. Logfiles are stored in **`rpt`**

- **`zcu102`**: it contains whatever needed to compile and run the application on ZCU102
  - **`baseline`**: application files of the baseline quantized CNN. Logfiles are stored in **`rpt`**
  - **`pruned`**: application files of the pruned and quantized CNN. Logfiles are stored in **`rpt`**
  - **`test_images`**: images used at run time to compute the top-5 predictions

Step1: decent_miniVggNet.sh script

```
#!/usr/bin/env bash

DNNDK_ROOT=/home/ML/DNNDK/tools

#working directory
work_dir=$(pwd)
#path of float model
model_dir=${work_dir}
#output directory
output_dir=${work_dir}/decent_output

#soft link to the calibration data
ln -s /home/ML/cifar10/input/cifar10_jpg/calib  /home/ML/cifar10/deephi/miniVggNet/quantiz/data/calib

# copy input files from miniVggNet Caffe project via soft links
ln -s /home/ML/cifar10/caffe/models/miniVggNet/m3/deephi_train_val_3_miniVggNet.prototxt /home/ML/cifar10/deephi/miniVggNet/quantiz/float.prototxt

ln -s /home/ML/cifar10/caffe/models/miniVggNet/m3/snapshot_3_miniVggNet__iter_40000.caffemodel /home/ML/cifar10/deephi/miniVggNet/quantiz/float.caffemodel


# run DECENT
$DNNDK_ROOT/decent     quantize                    \
            -model ${model_dir}/float.prototxt     \
            -weights ${model_dir}/float.caffemodel \
            -output_dir ${output_dir} \
            -method 1 \
```

Command:

`source decent_miniVggNet.sh 2>&1 | tee logfile_decent_miniVggNet.txt`

Step1: run decent (with auto_test) 1/2

Fragments of decent logfile:

- check CNN description and then run calibration

```
I0831 11:28:32.822849 28315 convert_proto.cpp:160] Opening file ./data/calib/calibration.txt
I0831 11:28:32.824931 28315 convert_proto.cpp:171] A total of 1000 images.
I0831 11:28:32.826131 28315 convert_proto.cpp:2286]  Merge InnerProductBatchNorm -> InnerProduct: fc1 + bn5
I0831 11:28:32.846849 28315 convert_proto.cpp:2286]  Merge InnerProductBatchNorm -> InnerProduct: fc1 + bn5
I0831 11:28:32.881103 28315 net.cpp:323] The NetState phase (0) differed from the phase (1) specified by a rule in layer data
I0831 11:28:32.881122 28315 net.cpp:323] The NetState phase (0) differed from the phase (1) specified by a rule in layer accuracy-top1
I0831 11:28:32.881125 28315 net.cpp:323] The NetState phase (0) differed from the phase (1) specified by a rule in layer accuracy-top5
I0831 11:28:32.881278 28315 net.cpp:52] Initializing net from parameters:
state {
  phase: TRAIN
}
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    crop_size: 32
    mean_value: 125
    mean_value: 123
    mean_value: 114
  }
  image_data_param {
    source: "./data/calib/calibration.txt"
    batch_size: 10
    shuffle: true
    root_folder: "./data/calib/"
  }
}
layer {
  name: "data_fixed"
  type: "FixedNeuron"
  bottom: "data"
  top: "data"
  param {
    lr_mult: 0
-:---  logfile_decent_miniVggNet_autotest.txt    1% L39    (Text)
```

Step1: run `decent` (with `auto_test`) 2/2

Fragments of decent logfile:

```
I0109 17:33:16.115645 14383 net_test.cpp:339] Test iter: 43/50, loss = 0.516615
I0109 17:33:16.115649 14383 net_test.cpp:339] Test iter: 43/50, top-1 = 0.82
I0109 17:33:16.115653 14383 net_test.cpp:339] Test iter: 43/50, top-5 = 0.98
I0109 17:33:16.122541 14383 net_test.cpp:339] Test iter: 44/50, accuracy = 0.84
I0109 17:33:16.122555 14383 net_test.cpp:339] Test iter: 44/50, loss = 0.443076
I0109 17:33:16.122558 14383 net_test.cpp:339] Test iter: 44/50, top-1 = 0.84
I0109 17:33:16.122561 14383 net_test.cpp:339] Test iter: 44/50, top-5 = 1
I0109 17:33:16.128968 14383 net_test.cpp:339] Test iter: 45/50, accuracy = 0.94
I0109 17:33:16.128983 14383 net_test.cpp:339] Test iter: 45/50, loss = 0.227701
I0109 17:33:16.128986 14383 net_test.cpp:339] Test iter: 45/50, top-1 = 0.94
I0109 17:33:16.128988 14383 net_test.cpp:339] Test iter: 45/50, top-5 = 1
I0109 17:33:16.135323 14383 net_test.cpp:339] Test iter: 46/50, accuracy = 0.9
I0109 17:33:16.135336 14383 net_test.cpp:339] Test iter: 46/50, loss = 0.289338
I0109 17:33:16.135340 14383 net_test.cpp:339] Test iter: 46/50, top-1 = 0.9
I0109 17:33:16.135344 14383 net_test.cpp:339] Test iter: 46/50, top-5 = 1
I0109 17:33:16.141512 14383 net_test.cpp:339] Test iter: 47/50, accuracy = 0.96
I0109 17:33:16.141526 14383 net_test.cpp:339] Test iter: 47/50, loss = 0.128769
I0109 17:33:16.141530 14383 net_test.cpp:339] Test iter: 47/50, top-1 = 0.96
I0109 17:33:16.141533 14383 net_test.cpp:339] Test iter: 47/50, top-5 = 1
I0109 17:33:16.147711 14383 net_test.cpp:339] Test iter: 48/50, accuracy = 0.96
I0109 17:33:16.147724 14383 net_test.cpp:339] Test iter: 48/50, loss = 0.175614
I0109 17:33:16.147728 14383 net_test.cpp:339] Test iter: 48/50, top-1 = 0.96
I0109 17:33:16.147730 14383 net_test.cpp:339] Test iter: 48/50, top-5 = 1
I0109 17:33:16.154798 14383 net_test.cpp:339] Test iter: 49/50, accuracy = 0.94
I0109 17:33:16.154812 14383 net_test.cpp:339] Test iter: 49/50, loss = 0.224946
I0109 17:33:16.154816 14383 net_test.cpp:339] Test iter: 49/50, top-1 = 0.94
I0109 17:33:16.154819 14383 net_test.cpp:339] Test iter: 49/50, top-5 = 1
I0109 17:33:16.161334 14383 net_test.cpp:339] Test iter: 50/50, accuracy = 0.86
I0109 17:33:16.161348 14383 net_test.cpp:339] Test iter: 50/50, loss = 0.50688
I0109 17:33:16.161351 14383 net_test.cpp:339] Test iter: 50/50, top-1 = 0.86
I0109 17:33:16.161355 14383 net_test.cpp:339] Test iter: 50/50, top-5 = 0.98
I0109 17:33:16.161357 14383 net_test.cpp:346] Test Results:
I0109 17:33:16.161360 14383 net_test.cpp:347] Loss: 0.383396
I0109 17:33:16.161363 14383 net_test.cpp:361] accuracy = 0.8744
I0109 17:33:16.161370 14383 net_test.cpp:361] loss = 0.383396 (* 1 = 0.383396 loss)
I0109 17:33:16.161373 14383 net_test.cpp:361] top-1 = 0.8744
I0109 17:33:16.161376 14383 net_test.cpp:361] top-5 = 0.9924
I0109 17:33:16.161379 14383 net_test.cpp:387] Test Done!
I0109 17:33:16.312860 14383 decent.cpp:333] Start Deploy
I0109 17:33:16.341810 14383 decent.cpp:341] Deploy Done!
----------------------------------------------------
Output Deploy Weights: "/home/ML/cifar10/deephi/miniVggNet/quantiz/decent_output/deploy.caffemodel"
Output Deploy Model:   "/home/ML/cifar10/deephi/miniVggNet/quantiz/decent_output/deploy.prototxt"
```

- Check accuracy on the quantized model: 0.87 (top-1) and 0.99 (top-5)

- Generate output files:
  - deploy.caffemodel
  - deploy.prototxt

Step1: prepare the input files 1/2

- This and next pages are only to explain how the input files should be prepared. Such changes were already saved in the file named **deephi_train_val_3_miniVggNet.prototxt**

  - From the trained model with the highest prediction accuracy copy the caffemodel and training description files and rename them. For example, assuming the best Caffe-trained original model is placed in `<wkr>/models/miniVggNet/m3` do the following:

```
cd <wkr>/models/miniVggNet/m3
ln -s <wkr>/models/miniVggNet/m3/train_val_3_miniVggNet.prototxt float.prototxt
ln -s <wkr>/models/miniVggNet/m3/snapshot_3_miniVggNet__iter_40000.caffemodel float.caffemodel
```

  - `float.prototxt` and `float.caffemodel` are the 2 input files to the quantization process (just renamed from the `m3` models originally trained in Caffe)

  - Now you need to do some changes to `float.prototxt` file (see next page):
    - Remove the **`Data`** type layers for the original TRAIN phase
    - Add an **`ImageData`** type layer with the calibration images for the new TRAIN phase
    - Add on the bottom two **`Accuracy`** type layers to compute top-1 and top-5 accuracies
    - Remove the mean file and put separate values (DPU does not support reading a mean file)

Step1: prepare the input files 2/2

add this new layer on top

comment this layer

add these 2 new layers at the bottom

```
# add below layer for Quantization with DeePhi' DECENT
################################################################################
# CIFAR10 miniVggNet m3
layer {
  name: "data"
  type: "ImageData"
  top: "data"
  top: "label"
  include {
    phase: TRAIN
  }
  transform_param {
    crop_size: 32
    mean_value: 125
    mean_value: 123
    mean_value: 114
    #mean file is not supported by DeePhi DPU
    #mean_file: "/home/danieleb/ML/cifar10/input/mean.binaryproto"
  }
  image_data_param {
    source: "./data/calib/calibration.txt"
    root_folder: "./data/calib/"
    batch_size: 10
    shuffle: true
  }
}
# comments below layer for the original TRAIN
################################################################################
#layer {
#  name: "data"
#  type: "Data"
#  top: "data"
#  top: "label"
#  include {
#   phase: TRAIN
#  }
#  transform_param {
#    mirror: true
#    #crop_size: 32
#    mean_file: "/home/danieleb/ML/cifar10/input/mean.binaryproto"
#  }
#  data_param {
#    source: "/home/danieleb/ML/cifar10/input/lmdb/train_lmdb"
#    batch_size: 128
#    backend: LMDB
#  }
#}
layer {
  name: "data"
  type: "Data"
  top: "data"
  top: "label"
  include {
    phase: TEST
  }
  transform_param {
    mirror: false
    crop_size: 32
    mean_value: 125
    mean_value: 123
    mean_value: 114
  }
  data_param {
    source: "/home/danieleb/ML/cifar10/input/lmdb/valid_lmdb"
  name: "fc2"
  type: "InnerProduct"
  bottom: "drop3"
  top: "fc2"
  param {
    lr_mult: 1
    decay_mult: 1
  }
  param {
    lr_mult: 2
    decay_mult: 0
  }
  inner_product_param {
    num_output: 10
    weight_filler {
      #type: "gaussian"
      type: "xavier"
      #std: 0.001
    }
    bias_filler {
      type: "constant"
      value: 1
    }
  }
}
layer {
  name: "loss"
  type: "SoftmaxWithLoss"
  bottom: "fc2"
  bottom: "label"
  top: "loss"
}
#layer {
#  name: "accuracy"
#  type: "Accuracy"
#  bottom: "fc2"
#  bottom: "label"
#  top: "accuracy"
#  include {
#    phase: TEST
#  }
#}
layer {
  name: "accuracy-top1"
  type: "Accuracy"
  bottom: "fc2"
  bottom: "label"
  top: "top-1"
  include {
    phase: TEST
  }
}
layer {
  name: "accuracy-top5"
  type: "Accuracy"
  bottom: "fc2"
  bottom: "label"
  top: "top-5"
  include {
    phase: TEST
  }
  accuracy_param {
    top_k: 5
  }
```

Step2: dncc_miniVggNet.sh script

```
#!/bin/bash

DNNDK_ROOT=/home/ML/DNNDK/tools

net=miniVggNet
model_dir=decent_output
output_dir=dnnc_output

echo "Compiling network: ${net}"

$DNNDK_ROOT/dnnc --prototxt=${model_dir}/deploy.prototxt     \
        --caffemodel=${model_dir}/deploy.caffemodel \
        --output_dir=${output_dir}                  \
        --net_name=${net}                           \
        --dpu=4096FA                                \
        --cpu_arch=arm64                            \
        --mode=debug                                \
        --save_kernel



echo " copying dpu elf file into /../zcu102/baseline/model/arm64_4096 "
cp ${output_dir}/dpu_${net}\_*.elf  ${output_dir}/../../zcu102/baseline/model/arm64_4096

echo " copying the test images to be used by the ZCU102"
cp -r /home/ML/cifar10/input/cifar10_jpg/test ${output_dir}/../../zcu102/test_images

```

Command:

```
source dnnc_miniVggNet.sh 2>&1 | tee logfile_dnnc_miniVggNet.txt
```

Step2: run dnnc

```
Compiling network: miniVggNet
[DNNC][Warning] Layer [loss] is not supported in DPU, deploy it in CPU instead.

DNNC Kernel Information

1. Overview
kernel numbers  : 2
kernel topology : 0 -> 1

2. Kernel Description in Detail
kernel id       : 0
kernel name     : miniVggNet_0
type            : DPUKernel (Supported, Running on DPU)
nodes           : NA
input node(s)   : conv1
output node(s)  : fc2

kernel id       : 1
kernel name     : miniVggNet_1
type            : CPUKernel (Not-Supported, Running on CPU)
nodes           : loss
input node(s)   : loss
output node(s)  : loss


-:**-  logfile_dnnc_miniVggNet.txt   All L3     (Text)
```

dnnc logfile

Steps 1 and 2: what we have done so far

- Running **`decent_miniVggNet.sh`** and **`dnnc_miniVggNet.sh`** scripts compile the miniVggNet model into DPU kernel, which is an ELF file containing the DPU instructions and parameters for the network model.
- The miniVggNet neural network model was divided as 2 different Kernels:
  - Kernel 0 : **`miniVggNet_0`** （running on DPU with ELF file dpu_miniVggNet_0.elf）
  - Kernel 1 : **`miniVggNet_1`** （running on ARM CPU）
- Kernel 0 runs all the layers of the CNN -i.e. CONV, BN, POOL, RELU, FC- on DPU in the PL of ZCU102 platform
- Kernel 1 run the SoftMax last layer on the PS of ZCU102 platform
- The quantized CNN has an accuracy unchanged from the original model (at least for this CNN case), as estimated by **decent**

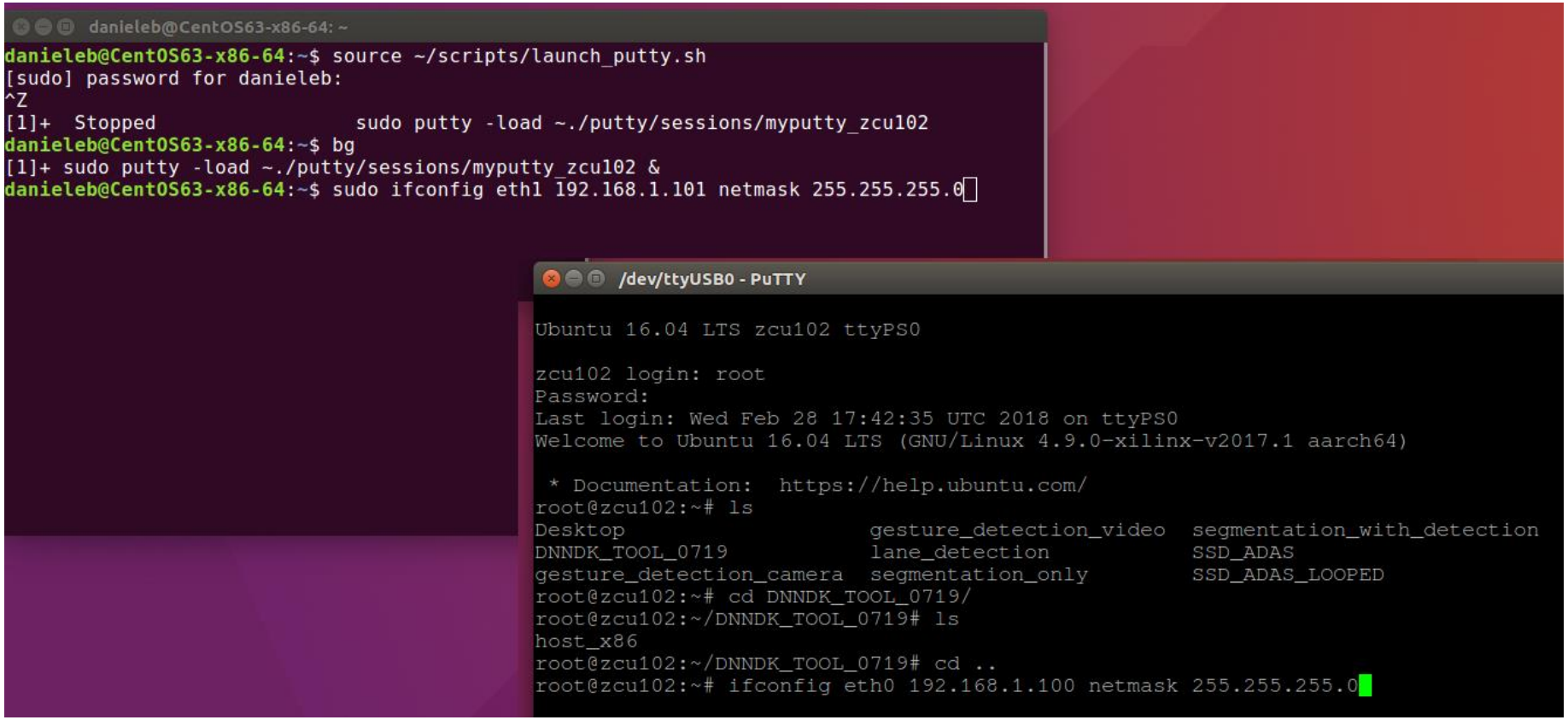
Step3: main.cc operations

- The main operations are:
  - 1. Call **`dpuOpen()`** to open DPU device
  - 2. Call **`dpuLoadKernel()`** to load DPU kernel miniVggNet_0 from miniVggNet model
  - 3. Call **`dpuCreateTask()`** to create task for each DPU kernel
  - 4. Coding “Softmax” CPU kernel to work with DPU kernel in **`run_miniVggNet()`** to do image classification
  - 5. Call **`dpuDestroyKernel()`** and **`dpuDestoyTask()`** to destroy Kernel and Task
  - 6. Call **`dpuClose()`** to close DPU device

Step3: run_miniVggNet() routine

- **`run_miniVggNet()`** does the following tasks:
  - 1. Read picture and set it as the input to DPU kernel miniVggNet_0 by calling **`dpuSetInputImage2()`** API
  - 2. Call **`dpuRunTask()`** to run taskConv convolution operation in minVggNet network model.
  - 3. Call **`dpuRunTask()`** to do full connection operation of taskFC on DPU.
  - 4. Do **Softmax** operation on CPU using the output of full connection operation as input.
  - 5. Output the **top-5 classification** category and the corresponding probability

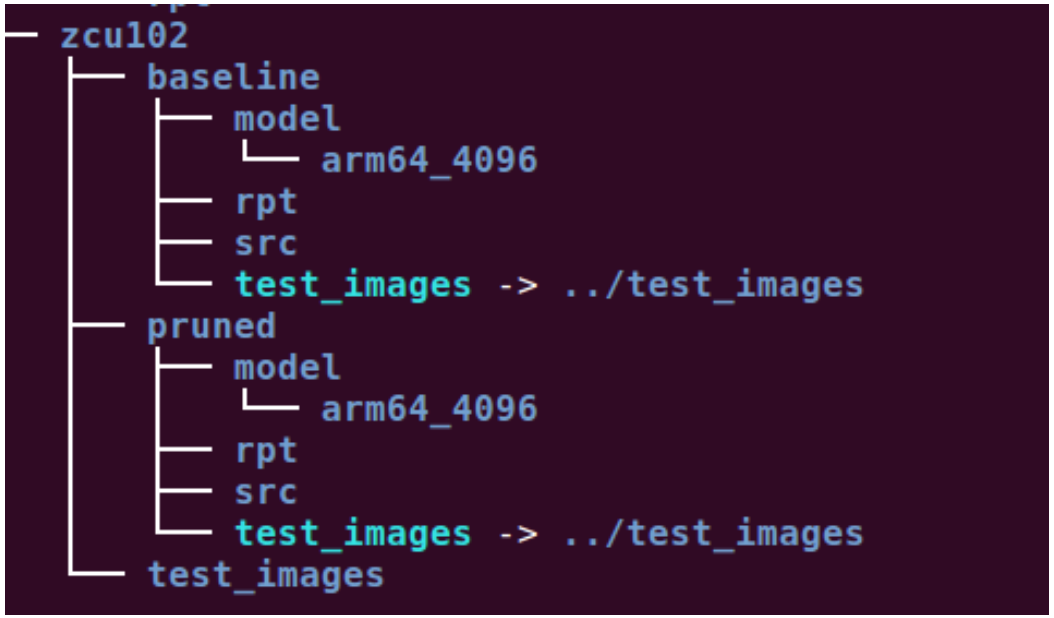
Step4: compile hybrid application on ZCU102 target



Copy the elf files related to the DPU kernels from Host PC to the Target board file system by **scp** command, having the Target ZCU102 board EP2P connected to Host PC:

- from Host, run `sudo ifconfig eth1 192.168.1.101 netmask 255.255.255.0`
- from Target, run `ifconfig eth0 192.168.1.100 netmask 255.255.255.0`

Steps 4 and 5: working on the ZCU102 board



I have created a tar file and copied it from/to host to/from target via **`scp`** command (see next pages)

Booth the ZCU102 board with DeePhi‘ SD-Card

I have created a folder named **`~/cifar10/miniVggNet/zcu102`** organized as shown on the left

The folder **`test_images`** has the same 1000 images of the CIFAR10 test folder (remember that during the training I used 50000 images of the train and 9000 images of the validation dataset)

The folder **`model/arm64_4096`** contains the same **`dpu_miniVggNet_0.elf`** kernel that was created during step2

Step5: run hybrid application on ZCU102 target

- To check the fps performance with multithreding I do the following commands:

```
source run_fps_miniVggNet.sh 2>&1 | tee ./rpt/logfile_fps_miniVggNet.txt
```

- To check the top-5 average accuracy I do the following commands:

```
source top5_miniVggNet 2>&1 | tee ./rpt/logfile_top5_miniVggNet.txt
```

Step5: best performance in fps

- Comment all unnecessary code with printf or file I/O
- Comment image resizing (the images are already 32x32)
- Comment top-5 computation
- **Best performance is 3799 with 4 threads**

```
12 make: warning:  Clock skew detected.  Your build may be incomplete.
13
14 ./miniVggNet 1
15 total image : 1000
16 [Time]695304us
17 [FPS]1438.22
18
19 ./miniVggNet 2
20 total image : 1000
21 [Time]390204us
22 [FPS]2562.76
23
24 ./miniVggNet 3
25 total image : 1000
26 [Time]271455us
27 [FPS]3683.85
28
29 ./miniVggNet 4
30 total image : 1000
31 [Time]263226us
32 [FPS]3799.02
33
34 ./miniVggNet 5
35 total image : 1000
36 [Time]276489us
37 [FPS]3616.78
38
39 ./miniVggNet 6
40 total image : 1000
41 [Time]281639us
42 [FPS]3550.64
```

Step5: compute top-5 accuracy

- Uncomment top-5 computation lines and add a line to print the image currently read just before the top-5 computation.
- See on the right the logfile captured at runtime
- You need a Python script to post process such logfile and compute the overall average accuracy at runtime (otherwise add some C code lines in the main) in order to compare it with your predictions from the original Caffe model

logfile_run_top5_miniVggNet.txt

```
total image : 1000
DBG imread ./images/frog_00001.jpg
[Top]0 prob = 0.940882  name = frog
[Top]1 prob = 0.028412  name = deer
[Top]2 prob = 0.028412  name = cat
[Top]3 prob = 0.001415  name = bird
[Top]4 prob = 0.000858  name = dog
DBG imread ./images/frog_00017.jpg
[Top]0 prob = 0.995822  name = frog
[Top]1 prob = 0.003169  name = cat
[Top]2 prob = 0.000908  name = dog
[Top]3 prob = 0.000045  name = horse
[Top]4 prob = 0.000035  name = ship
DBG imread ./images/horse_00007.jpg
[Top]0 prob = 0.923117  name = horse
[Top]1 prob = 0.045959  name = dog
[Top]2 prob = 0.027876  name = deer
[Top]3 prob = 0.001782  name = bird
[Top]4 prob = 0.000842  name = cat
DBG imread ./images/truck_00074.jpg
[Top]0 prob = 0.997527  name = truck
[Top]1 prob = 0.002473  name = automobile
[Top]2 prob = 0.000000  name = ship
[Top]3 prob = 0.000000  name = frog
[Top]4 prob = 0.000000  name = airplane
DBG imread ./images/frog_00092.jpg
[Top]0 prob = 0.999935  name = frog
[Top]1 prob = 0.000035  name = cat
[Top]2 prob = 0.000010  name = dog
[Top]3 prob = 0.000008  name = deer
[Top]4 prob = 0.000006  name = bird
DBG imread ./images/dog_00070.jpg
[Top]0 prob = 0.879771  name = dog
[Top]1 prob = 0.119064  name = cat
[Top]2 prob = 0.000625  name = frog
[Top]3 prob = 0.000295  name = bird
[Top]4 prob = 0.000230  name = deer
DBG imread ./images/bird_00057.jpg
[Top]0 prob = 0.603890  name = bird
```

Step5: compute top-5 accuracy

```
void TopK(const float *d, int size, int k, vector<string> &vkind) {
    assert(d && size > 0 && k > 0);
    priority_queue<pair<float, int>> q;

    for (auto i = 0; i < size; ++i) {
        q.push(pair<float, int>(d[i], i));
    }

    for (auto i = 0; i < k; ++i)
    {
      pair<float, int> ki = q.top();
      printf("[Top]%d prob = %-8f  name = %s\n", i, d[ki.second], vkind[ki.second].c_str());
      q.pop();
    }
}
```

```
-:---  main.cc        57% L174   (C++/l Abbrev)
        DPUTask *taskconv = dpuCreateTask(kernelconv, DPU_MODE_NORMAL); // profiling not enabled
        //DPUTask *taskconv = dpuCreateTask(kernelconv, DPU_MODE_PROF); // profiling enabled
        //enable profiling
        //int res1 = dpuEnableTaskProfile(taskconv);
        //if (res1!=0) printf("ERROR IN ENABLING TASK PROFILING FOR CONV KERNEL\n");

        for(unsigned int ind = i  ;ind < images.size();ind+=threadnum)
        {
            Mat img = imread(baseImagePath + images.at(ind));
            cout << "DBG imread " << baseImagePath + images.at(ind) << endl;
            //Size sz(32,32);
            //Mat img2; resize(img, img2, sz); //DB
            //run_miniVggNet(taskconv,img2); //DB: images are already 32x32 and do not need any resize
            run_miniVggNet(taskconv,img);
        }

        // Destroy DPU Tasks & free resources
        dpuDestroyTask(taskconv);
    });
-:---  main.cc        81% L258   (C++/l Abbrev)
```

Check the top-5 accuracy with a python script

- By capturing the output of DPU at runtime on a logfile, I can post-process it with a python script to check the effective top-1 and top-5 accuracies of quantized net:
- **top-1: 87% average accuracy measured at runtime, as also estimated by decent during quantization step1**

```
LINE:  airplane_00093.jpg

PREDICTED:  [ 3.] cat
EXPECTED :  [ 0.] airplane
[Top]0 prob = 0.925463  name = cat
[Top]1 prob = 0.035884  name = deer
[Top]2 prob = 0.021765  name = airplane
[Top]3 prob = 0.006236  name = bird
[Top]4 prob = 0.003782  name = horse


LINE:  truck_00028.jpg

PREDICTED:  [ 8.] ship
EXPECTED :  [ 9.] truck
[Top]0 prob = 0.999815  name = ship
[Top]1 prob = 0.000075  name = truck
[Top]2 prob = 0.000058  name = airplane
[Top]3 prob = 0.000021  name = frog
[Top]4 prob = 0.000021  name = automobile


LINE:  dog_00100.jpg

PREDICTED:  [ 2.] bird
EXPECTED :  [ 5.] dog
[Top]0 prob = 0.752371  name = bird
[Top]1 prob = 0.130742  name = dog
[Top]2 prob = 0.101822  name = cat
[Top]3 prob = 0.010732  name = frog
[Top]4 prob = 0.003075  name = deer


number of total images predicted  999
number of top1 false predictions  132
number of top1 right predictions  867
number of top5 false predictions  2
number of top5 right predictions  997
top1 accuracy = 0.87
top5 accuracy = 1.00
```

-:--- logfile_check_dpu_top1_10Kimages.txt Bot L14365 (Text)

```
top1_false = 0
top5_true  = 0
top5_false = 0
img_count  = 0
false_pred = 0

test_ids = np.zeros(([NUMEL,1]))
preds    = np.zeros(([NUMEL, 1]))
idx = 0

for ln in range(0, tot_lines):

    if "DBG" in lines[ln]:

        top5_lines = lines[ln:ln+6]
        s2 = top5_lines[0].index("_")
        class_name = top5_lines[0][20:s2].strip()
        #print 'DBG: found class ', class_name, ' in line ', ln, ': ', lines[ln]

        predicted = top5_lines[1][30 : ].strip()

        if class_name in top5_lines[1]:
            top1_true += 1
            top5_true += 1
        elif class_name in top5_lines[2]:
            top5_true += 1
            top1_false +=1
        elif class_name in top5_lines[3]:
            top5_true += 1
            top1_false +=1
        elif class_name in top5_lines[4]:
            top5_true += 1
            top1_false +=1
        elif class_name in top5_lines[5]:
            top5_true += 1
            top1_false +=1
        else:
            top5_false += 1
            top1_false +=1

        test_ids[idx] = labelNames[class_name] # ground truth
        preds[idx]    = labelNames[predicted ] # actual prediction

        if (predicted != class_name) :
            print "LINE: ", top5_lines[0].split("./")[1].strip()
            print "PREDICTED: ", preds[idx], predicted
            print "EXPECTED : ", test_ids[idx], class_name
            for k in range(1, 6):
                print top5_lines[k].strip()
            print "\n"


        img_count +=1
        idx += 1

        if ( idx == (NUMEL-1) ):
            break
```

-:--- check_dpu_runtime_accuracy.py 40% L109 (Python)