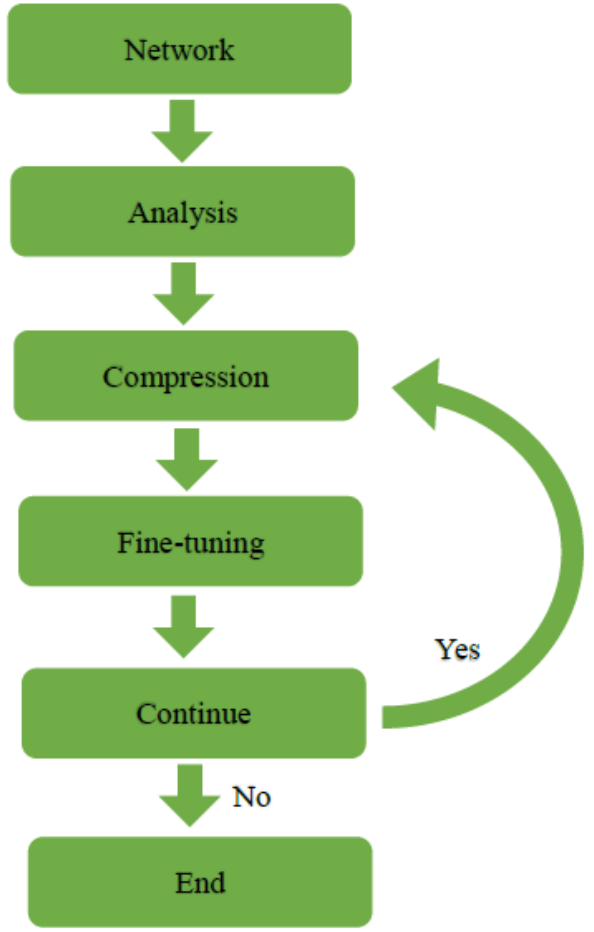
DeePhi’ ML Pruning

CNN Pruning (or Compression)

- Neural network pruning is an old idea, dating from 1990 (Yan Lecun‘s optimal braind damage work), where some of the network parameters are redundant and can be removed without effecting the neural network accuracy
- Weight pruning can be employed to any layer type, but it is most succesfull if it is applied to convolutional and fully-connected layers
- There are two benefits of pruning:
  - **Reduction of model size** – by removing some of the weights or of the channels, it is possible to efficiently compress this sparse CNN representation, therefore significantly reducing memory space required for CNN storage
  - **Increase in instance processing speed** – since after pruning significant number of weights or of channels will be zero, if the underlying architecture takes advantage of this, significant improvement in instance processing speed is possible
- All CNNs pruning algorithms can be divided into two classes:
  - Coarse-grained pruning – remove complete 3D kernels and/or feature map channels
  - Fine-grained pruning – only specific coefficients from 3D kernels are removed, based on some selection criteria. Each 3D kernel can have its coefficients removed from different locations

DeePhi' Pruning Overview

- Pruning is composed of the following functions:
  - **Analysis (*ana*):** Analyzing the model to find the pruning strategy.
  - **Compression (*compress*):** Compressing the model.
  - **Fine-tuning (*finetune*):** Fine-tuning the compressed model to improve model's performance.
  - **Transformation (*transform*):** Transform the compressed model to generate the final model.
- The process should be done iteratively, so that at the end of each round the model gets smaller and smaller
- You should use the same solver of the original Caffe training
- Pruning can require N x T time
  - with N the amount of pruning rounds (by 0.1 compression steps)
  - and T the time needed for the original Caffe training



Pruning and Quantization

- Remember that Pruning is just an optimization technique to reduce the complexity of the CNN in its original floating point model.
  - The DeePhi pruning process reduces the amount of features maps between layers.
- Once the CNN has been pruned, you need to Quantize it in order to be able to implement it on FPGA.
  - CNN designs on FPGAs are more efficient if done in fixed point arithmetic (i.e. 8-bit)
- Hence, after Pruning process you have to run a Quantization process !

Pruning of miniVggNet

Prepare the input files for pruning

- You need to have the following files in the working directory **`miniVggNet/pruning`**:
- 1) **`config.prototxt`** (see next pages)
- 2) **`solver.prototxt`**
  - the same solver of your original Caffe model just renamed, for example the same **`solver_3_miniVggNet.prototxt`** already used in previous Part 3a.
  - Just change the pathnames inside it!
- 3) **`train_val.prototxt`**
  - the same description file of your original Caffe model just renamed, for example the same **`train_val_3_miniVggNet.prototxt`** already used in previous Part 3a.
  - You need to add top-1 and top-5 accuracy layers at the end of it
- 4) **`float.caffemodel`**
  - the same weights file of your original Caffe model just renamed, for example the same **`snapshot_3_miniVggNet__iter_40000.caffemodel`** already used in previous Part 3a

train_val.prototxt

```
#################################################################################
name: "miniVggNet on Cifar10 m3"
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_file: "/home/danieleb/ML/cifar10/input/mean.binaryproto"
  }
  data_param {
    source: "/home/danieleb/ML/cifar10/input/lmdb/valid_lmdb"
    batch_size: 50
    backend: LMDB
  }
}

######## CONV1=>BN1=>RELU1=>CONV2=>BN2=>RELU2=>POOL1=>DROP1
#note that BN + Scale + ReLU the all use InPlace

layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"
  # learning rate and decay multipliers for the filters
    param {
-:---  train_val.prototxt   Top L1     (Fundamental)
```

```
    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-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
  }
}
-:---  train_val.prototxt   Bot L499   (Fundamental)
```

add these 2 new layers at the bottom

XILINX.

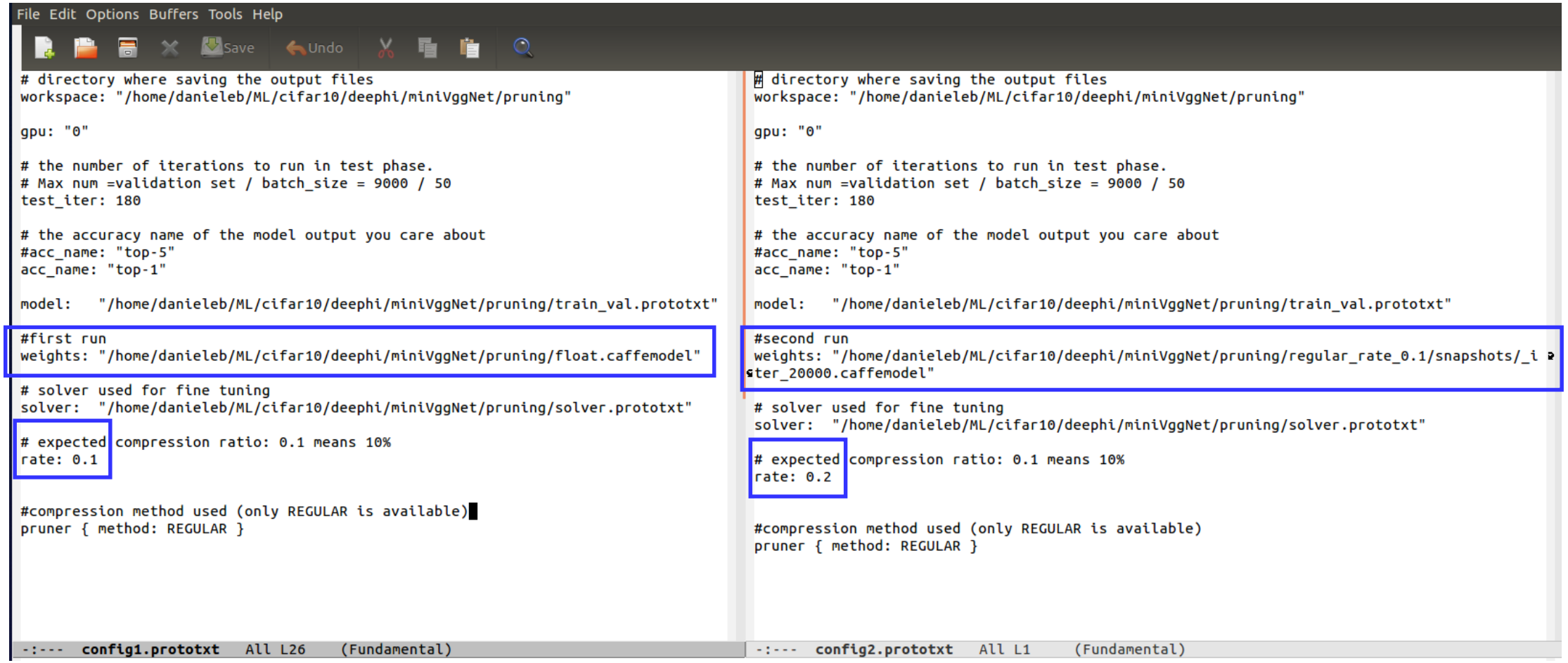
Pruning design flow: 7 rounds for miniVggNet

```
# directory where saving the output files
workspace: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning"

gpu: "0"

# the number of iterations to run in test phase: Max num =validation set / batch_size = 9000 / 50
test_iter: 180

# the accuracy name of the model output you care about
acc_name: "top-1"

# model for the training
model:   "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/train_val.prototxt"

## first run
weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/float.caffemodel"

## second run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.1/snapshots/_iter_20000.caffemodel"
## third run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.2/snapshots/_iter_20000.caffemodel"
## 4-th run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.3/snapshots/_iter_20000.caffemodel"
## 5-th run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.4/snapshots/_iter_20000.caffemodel"
## 6-th run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.5/snapshots/_iter_20000.caffemodel"
## 7-th run
#weights: "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.6/snapshots/_iter_20000.caffemodel"

# solver used for fine tuning
solver:  "/home/danieleb/ML/cifar10/deephi/miniVggNet/pruning/solver.prototxt"

# expected compression ratio: 0.1 means 10%
rate: 0.1  #first run
#rate: 0.2 #second run
#rate: 0.3 #third run
#rate: 0.4 # 4-th run
#rate: 0.5 # 5-th run
#rate: 0.6 # 6-th run
#rate: 0.7 # 7-th run

#compression method used (only REGULAR is available)
pruner { method: REGULAR }
```

Instead of editing any time the **`config.prototxt`** file, I have created 7 different copies of it (see next 2 pages) and then I call them from the shell script named **`pruning_flow.sh`** with the command

```
source pruning_flow.sh 2>&1 | tee logfile_whole_pruning_flow_miniVggNet.txt
```

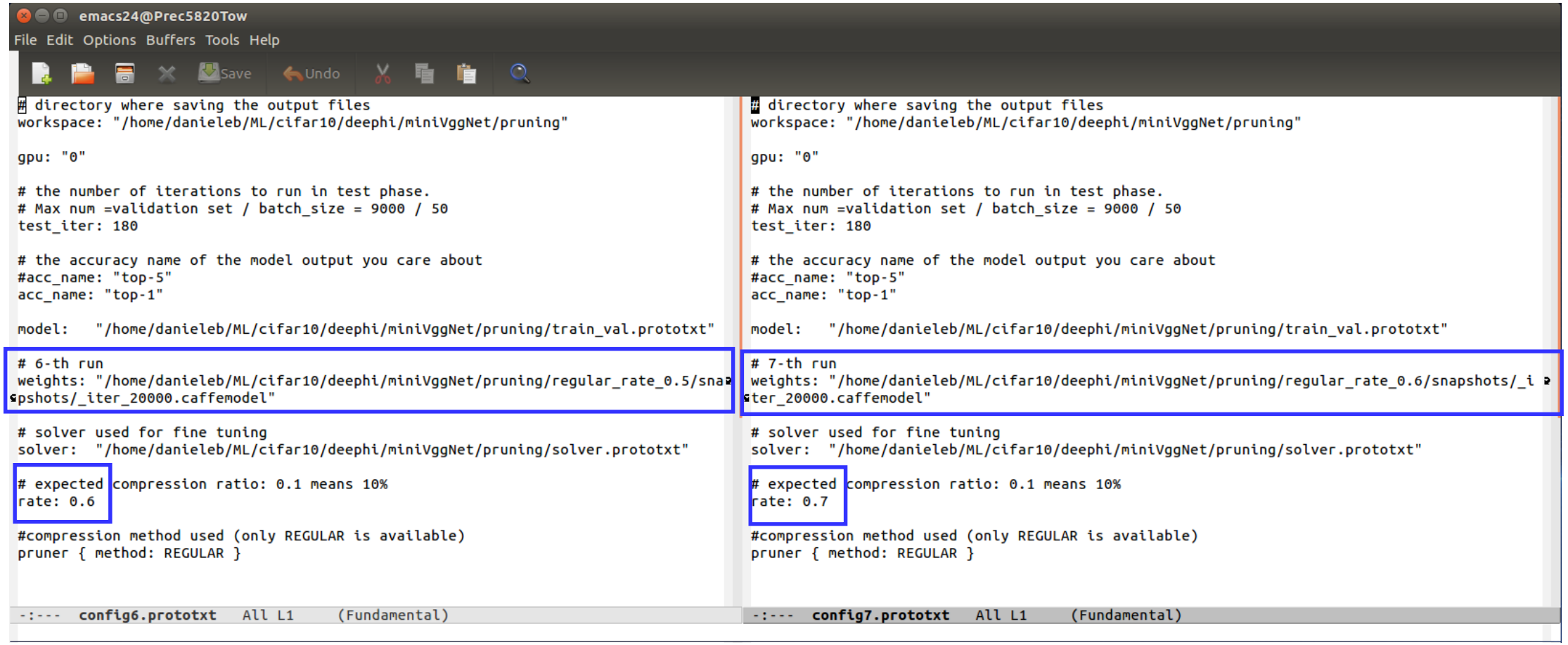
Configuration files (1st and 2nd rounds)



config1.prototxt

config2.prototxt

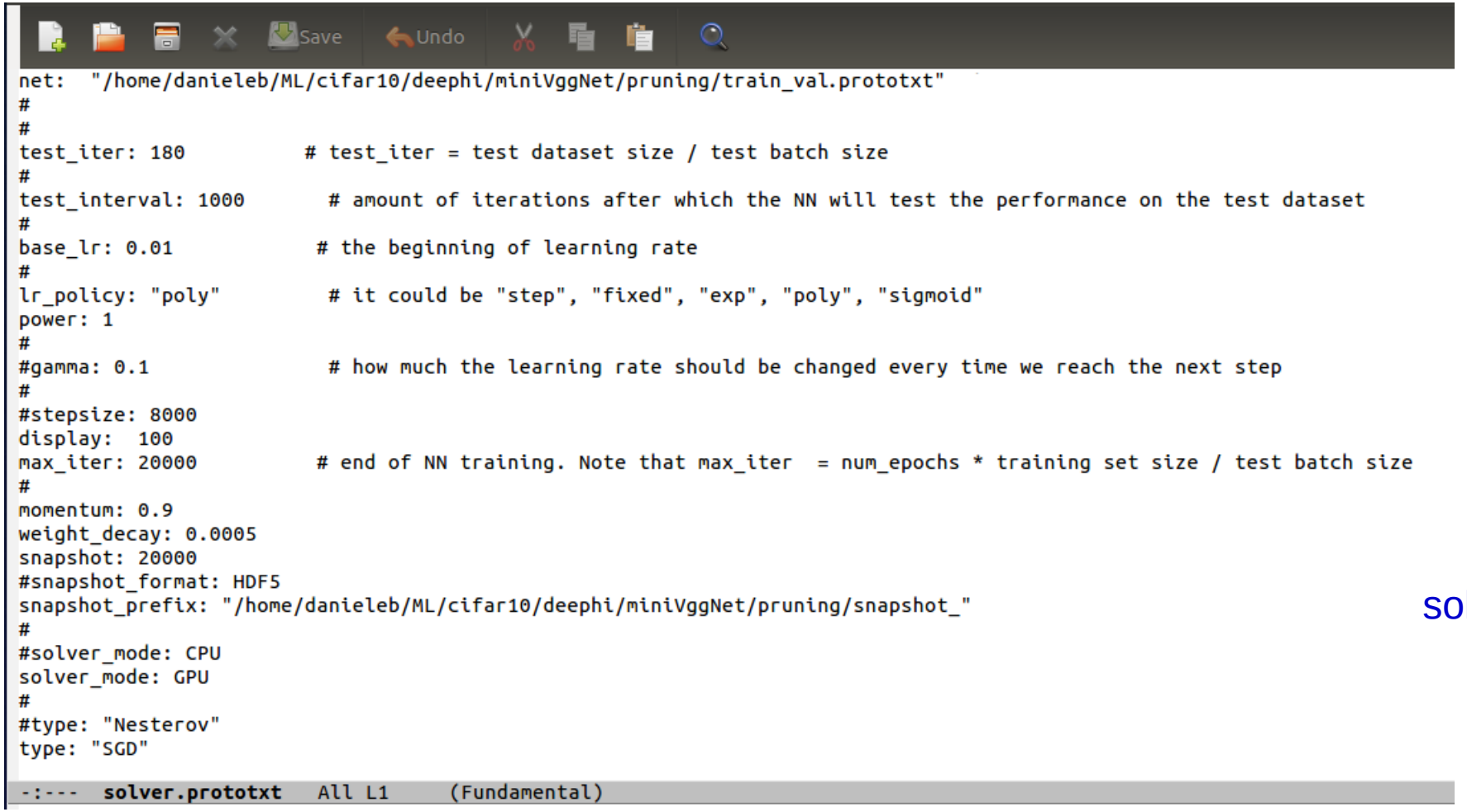
Configuration files (6-th and 7-th run)



config6.prototxt

config7.prototxt

solver.prototxt



solver.prototxt

Pruning process: results

- After round 1: -09.7% less operations and -08.5% less weights
- After round 2: -17.1% less operations and -11.1% less weights
- After round 3: -25.3% less operations and -22.7% less weights
- After round 4: -39.9% less operations and -34.9% less weights
- After round 5: -49.4% less operations and -42.4% less weights
- After round 6: -58.1% less operations and -52.3% less weights
- After round 7: -68.2% less operations and -64.6% less weights

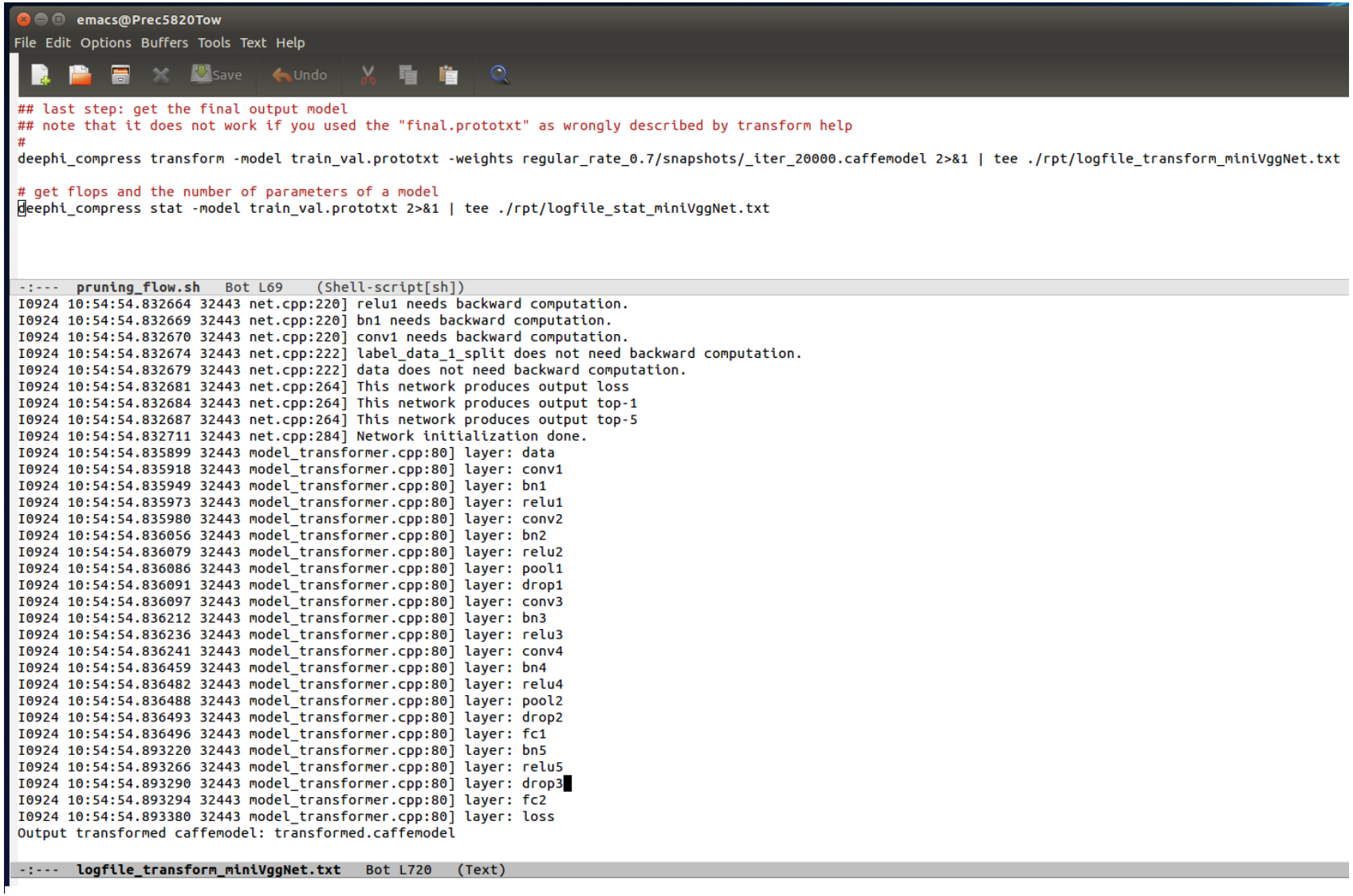
Pruning process: finetune vs. compress

```
emacs@Prec5820Tow
1252 I0109 18:07:11.879621 24771 caffe_interface.cpp:125] Batch 179, top-5 = 0.84
1253 I0109 18:07:11.879622 24771 caffe_interface.cpp:130] Loss: 2.06076
1254 I0109 18:07:11.879629 24771 caffe_interface.cpp:142] loss = 2.06076 (* 1 = 2.06076 loss)
1255 I0109 18:07:11.879633 24771 caffe_interface.cpp:142] top-1 = 0.527778
1256 I0109 18:07:11.879638 24771 caffe_interface.cpp:142] top-5 = 0.921111
1257 I0109 18:07:12.028092 24771 pruning_runner.cpp:306] pruning done, output model: /home/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.7/sparse.caffemode
1258 I0109 18:07:12.028126 24771 pruning_runner.cpp:320] summary of REGULAR compression with rate 0.7:
1259 +-------------------------------------------------------------+
1260 | Item           | Baseline       | Compressed     | Delta     |
1261 +-------------------------------------------------------------+
1262 | Accuracy       | 0.86533314     | 0.527777791    | -0.337555349   |
1263 +-------------------------------------------------------------+
1264 | Weights        | 68389          | 25157          | -63.2148438%   |
1265 +-------------------------------------------------------------+
1266 | Operations     | 49053696       | 15276032       | -68.858551%    |
1267 +-------------------------------------------------------------+
1268 To fine-tune the compressed model, please run:
1269 deephi_compress finetune -config config7.prototxt
-:---  logfile_compress7_miniVggNet.txt   98% L1257  (Text)
2087 I0109 18:10:08.489218 26013 sgd_solver.cpp:106] Iteration 19800, lr = 9.99999e-05
2088 I0109 18:10:09.351186 26013 solver.cpp:266] Iteration 19900 (116.018 iter/s, 0.861935s/100 iter), loss = 0.233155
2089 I0109 18:10:09.351212 26013 solver.cpp:285]     Train net output #0: loss = 0.233155 (* 1 = 0.233155 loss)
2090 I0109 18:10:09.351218 26013 sgd_solver.cpp:106] Iteration 19900, lr = 5e-05
2091 I0109 18:10:10.206316 26013 solver.cpp:929] Snapshotting to binary proto file /home/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.7/snapshots/_iter_20000.caffemodel
2092 I0109 18:10:10.272621 26013 sgd_solver.cpp:273] Snapshotting solver state to binary proto file /home/ML/cifar10/deephi/miniVggNet/pruning/regular_rate_0.7/snapshots/_iter_2
2093 I0109 18:10:10.285486 26013 solver.cpp:378] Iteration 20000, loss = 0.0479589
2094 I0109 18:10:10.285509 26013 solver.cpp:418] Iteration 20000, Testing net (#0)
2095 I0109 18:10:10.502454 26013 solver.cpp:517]     Test net output #0: loss = 0.461745 (* 1 = 0.461745 loss)
2096 I0109 18:10:10.502470 26013 solver.cpp:517]     Test net output #1: top-1 = 0.851111
2097 I0109 18:10:10.502473 26013 solver.cpp:517]     Test net output #2: top-5 = 0.991889
2098 I0109 18:10:10.502477 26013 solver.cpp:386] Optimization Done (113.517 iter/s).
2099 I0109 18:10:10.502482 26013 caffe_interface.cpp:530] Optimization Done.
-:---  logfile_finetune7_miniVggNet.txt   Bot L2100  (Text)
```

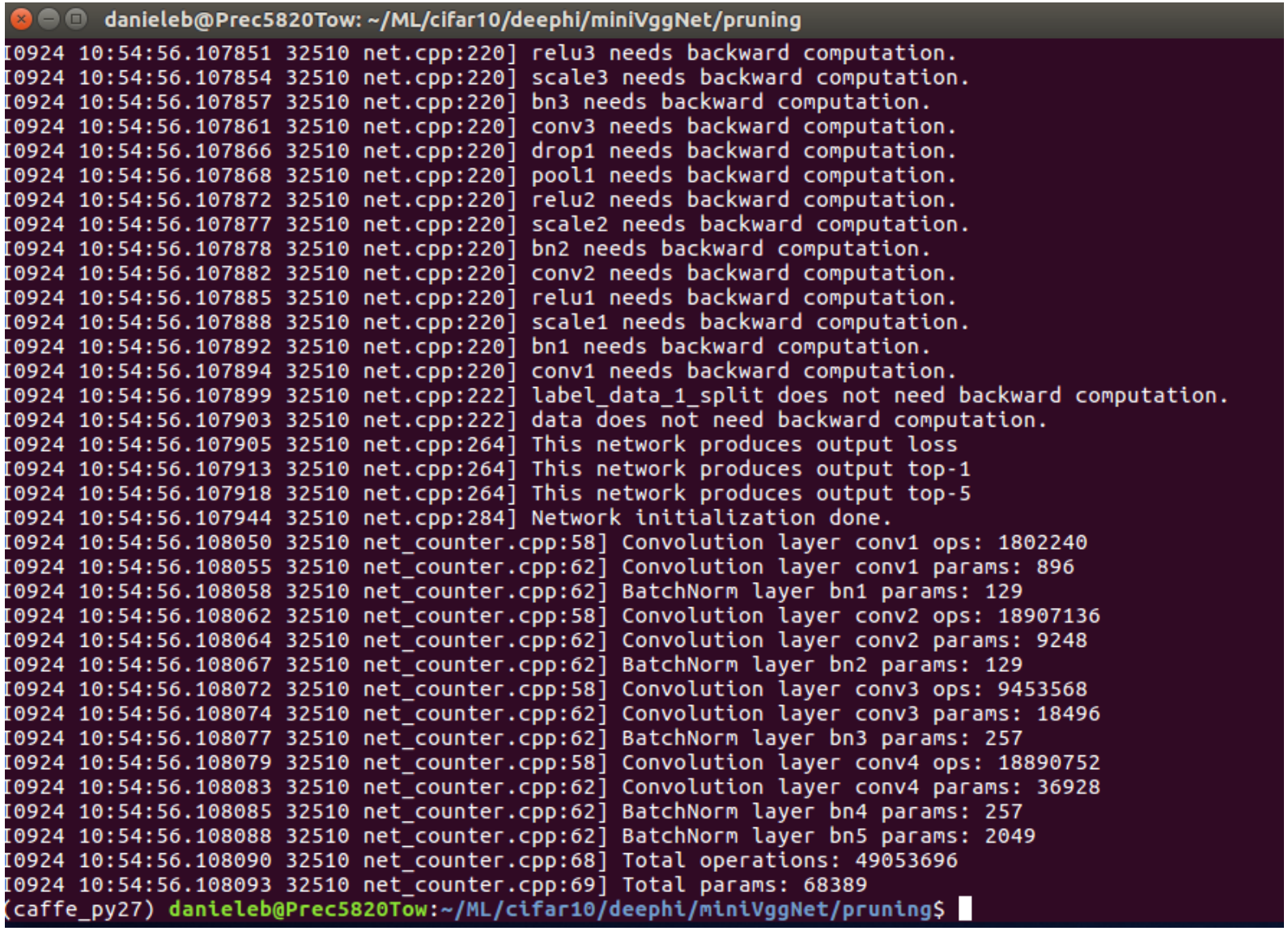
Pruning process: finetune vs. compress

- **Finetune** is the necessary step to recover the accuracy delta achieved after the **compress** step
- In the previous page, after the **compress** step, accuracy drops from 86% (baseline) to 52% (round 7 of pruning process)
- But after **finetune**, it come back to 85% The effective accuracy delta is so less than 2%
- Finetune is a real effective re-training and might requires as many iterations as the original Caffe training.
  - in the miniVggNet example I used only 20000 iterations of finetune instead of the 40000 iterations of the original training to save processing time, with more iterations results should become even better

The pruning step: Transform



The pruning step: statistics



Pruning: what we have done so far

- Starting from the input files we have generated 2 output files:
  - input files: `float.caffemodel, float.prototxt, solver.prototxt, train_val.prototxt`
  - 7 rounds of compress and finetune with half the iterations (20000) of the original Caffe training (40000)
  - output files: `transformed.caffemodel, final.prototxt`
- The compressed net has now -68% less operations and -63% less weights than the original baseline net
- The accuracy has drop from 87% (baseline) to 85% (compressed)

Quantizing the pruned CNN

- Now you can run Deephi' Quantization on the `final.prototxt` and `transformed.caffemodel`, exactly as described in previous pages
- Remember to edit the `final.prototxt` file in order to add the calibration images during TRAIN phase instead of the LMDB database and to add the mean values.
- Once edited, I rename it `q_float.prototxt`

Run Quantization: decent script

```
source decent_pruned_miniVggNet.sh 2>&1 | tee ./rpt/logfile_decent_pruned_miniVggNet.txt
```

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

#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/pruning/quantiz/data/calib

# next commented 2 lines are only for documentation
## cp ${model_dir}/regular_rate_0.7/final.prototxt ${model_dir}/quantiz/q_final.prototxt
## then edit it to add the calibration images

# run DECENT
decent     quantize                                    \
       -model ${model_dir}/q_final.prototxt \
       -weights ${model_dir}/../transformed.caffemodel \
       -output_dir ${output_dir} \
       -method 1 \
       -auto_test -test_iter 50
```

Run Quantization: dnnc script

```
source dnnc_pruned_miniVggNet.sh 2>&1 | tee ./rpt/logfile_dnnc_pruned_miniVggNet.txt
```

```
#!/bin/bash

net=miniVggNet
model_dir=decent_output
output_dir=dnnc_output

echo "Compiling network: ${net}"

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/pruned/model/arm64_4096 "
cp ${output_dir}/dpu_${net}\_*.elf  ${output_dir}/../../../zcu102/pruned/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
cd /home/ML/cifar10/deephi/miniVggNet/zcu102/pruned
```

Final fragment of decent logfile: top-1 accuracy 85.7%

```
1997 I0110 09:43:53.641299 34753 net_test.cpp:339] Test iter: 43/50, top-5 = 1
1998 I0110 09:43:53.646556 34753 net_test.cpp:339] Test iter: 44/50, loss = 0.479192
1999 I0110 09:43:53.646570 34753 net_test.cpp:339] Test iter: 44/50, top-1 = 0.86
2000 I0110 09:43:53.646574 34753 net_test.cpp:339] Test iter: 44/50, top-5 = 1
2001 I0110 09:43:53.651903 34753 net_test.cpp:339] Test iter: 45/50, loss = 0.428709
2002 I0110 09:43:53.651917 34753 net_test.cpp:339] Test iter: 45/50, top-1 = 0.8
2003 I0110 09:43:53.651921 34753 net_test.cpp:339] Test iter: 45/50, top-5 = 1
2004 I0110 09:43:53.657212 34753 net_test.cpp:339] Test iter: 46/50, loss = 0.374592
2005 I0110 09:43:53.657227 34753 net_test.cpp:339] Test iter: 46/50, top-1 = 0.88
2006 I0110 09:43:53.657230 34753 net_test.cpp:339] Test iter: 46/50, top-5 = 0.98
2007 I0110 09:43:53.663606 34753 net_test.cpp:339] Test iter: 47/50, loss = 0.334885
2008 I0110 09:43:53.663620 34753 net_test.cpp:339] Test iter: 47/50, top-1 = 0.9
2009 I0110 09:43:53.663625 34753 net_test.cpp:339] Test iter: 47/50, top-5 = 1
2010 I0110 09:43:53.669240 34753 net_test.cpp:339] Test iter: 48/50, loss = 0.231453
2011 I0110 09:43:53.669255 34753 net_test.cpp:339] Test iter: 48/50, top-1 = 0.94
2012 I0110 09:43:53.669258 34753 net_test.cpp:339] Test iter: 48/50, top-5 = 1
2013 I0110 09:43:53.674574 34753 net_test.cpp:339] Test iter: 49/50, loss = 0.242736
2014 I0110 09:43:53.674588 34753 net_test.cpp:339] Test iter: 49/50, top-1 = 0.96
2015 I0110 09:43:53.674592 34753 net_test.cpp:339] Test iter: 49/50, top-5 = 1
2016 I0110 09:43:53.679831 34753 net_test.cpp:339] Test iter: 50/50, loss = 0.682052
2017 I0110 09:43:53.679843 34753 net_test.cpp:339] Test iter: 50/50, top-1 = 0.84
2018 I0110 09:43:53.679847 34753 net_test.cpp:339] Test iter: 50/50, top-5 = 0.98
2019 I0110 09:43:53.679850 34753 net_test.cpp:346] Test Results:
2020 I0110 09:43:53.679852 34753 net_test.cpp:347] Loss: 0.453493
2021 I0110 09:43:53.679858 34753 net_test.cpp:361] loss = 0.453493 (* 1 = 0.453493 loss)
2022 I0110 09:43:53.679862 34753 net_test.cpp:361] top-1 = 0.8572
2023 I0110 09:43:53.679865 34753 net_test.cpp:361] top-5 = 0.9928
2024 I0110 09:43:53.679868 34753 net_test.cpp:387] Test Done!
2025 I0110 09:43:53.829368 34753 decent.cpp:333] Start Deploy
2026 I0110 09:43:53.855823 34753 decent.cpp:341] Deploy Done!
2027 ----------------------------------------------------
2028 Output Deploy Weights: "/home/ML/cifar10/deephi/miniVggNet/pruning/quantiz/decent_output/deploy.caffemodel"
2029 Output Deploy Model:   "/home/ML/cifar10/deephi/miniVggNet/pruning/quantiz/decent_output/deploy.prototxt"
```

Fragment of dnnc logfile

```
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

 copying dpu elf file into ../../zcu102/pruned/model/arm64_4096
 copying the test images to be used by the ZCU102
```

```
-:---  logfile_dnnc_pruned_miniVggNet.txt   All L27    (Text)
```

Run time execution on ZCU102: fps

```
./miniVggNet 1
total image : 1000
[Time]569072us
[FPS]1757.25

./miniVggNet 2
total image : 1000
[Time]290007us
[FPS]3448.19

./miniVggNet 3
total image : 1000
[Time]201443us
[FPS]4964.18

./miniVggNet 4
total image : 1000
[Time]187942us
[FPS]5320.79

./miniVggNet 5
total image : 1000
[Time]204630us
[FPS]4886.87

./miniVggNet 6
total image : 1000
[Time]200035us
[FPS]4999.13
```

- Best performance in terms of frames per sec (fps) with 4 threads: 5320 fps vs. the 3799 of baseline net

Commands on ZCU102 target board:

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

source run_top5_miniVggNet.sh 2>&1 | tee \
  ./rpt/logfile_top5_pruned_miniVggNet.txt
```

Run time execution on ZCU102: top-1 accuracy

- 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 pruned and quantized net:
- **top-1: 84% measured at runtime vs. 85.7% expected/estimated by decent**
- **decent also allows re-training the quantized CNN, but this is beyond the scope of this tutorial..**

```
LINE:  bird_00026.jpg

PREDICTED:  [7.] horse
EXPECTED :  [2.] bird
[Top]0 prob = 0.830710  name = horse
[Top]1 prob = 0.087556  name = bird
[Top]2 prob = 0.041359  name = deer
[Top]3 prob = 0.025085  name = dog
[Top]4 prob = 0.015215  name = cat


LINE:  automobile_00066.jpg

PREDICTED:  [4.] deer
EXPECTED :  [1.] automobile
[Top]0 prob = 0.718542  name = deer
[Top]1 prob = 0.097244  name = bird
[Top]2 prob = 0.058981  name = truck
[Top]3 prob = 0.058981  name = cat
[Top]4 prob = 0.021698  name = ship


LINE:  bird_00093.jpg

PREDICTED:  [6.] frog
EXPECTED :  [2.] bird
[Top]0 prob = 0.421035  name = frog
[Top]1 prob = 0.255371  name = deer
[Top]2 prob = 0.154890  name = cat
[Top]3 prob = 0.154890  name = bird
[Top]4 prob = 0.012714  name = dog


number of total images predicted  999
number of top1 false predictions  161
number of top1 right predictions  838
number of top5 false predictions  6
number of top5 right predictions  993
top1 accuracy = 0.84
top5 accuracy = 0.99
-:---  logfile_check_dpu_top5_pruned_miniVggNet.txt
```