ECP5 Face Identification Quick Start Guide

Application Note

FPGA-AN-02010-1.0

October 2019

Disclaimers

Lattice makes no warranty, representation, or guarantee regarding the accuracy of information contained in this document or the suitability of its products for any particular purpose. All information herein is provided AS IS and with all faults, and all risk associated with such information is entirely with Buyer. Buyer shall not rely on any data and performance specifications or parameters provided herein. Products sold by Lattice have been subject to limited testing and it is the Buyer's responsibility to independently determine the suitability of any products and to test and verify the same. No Lattice products should be used in conjunction with mission- or safety-critical or any other application in which the failure of Lattice’s product could create a situation where personal injury, death, severe property or environmental damage may occur. The information provided in this document is proprietary to Lattice Semiconductor, and Lattice reserves the right to make any changes to the information in this document or to any products at any time without notice.

Contents

Figures

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CKPT	Checkpoint
FPGA	Field-Programmable Gate Array

1. Introduction

This document provides a quick guide on how to train a machine and create a frozen file for the Lattice Machine Learning development using the Lattice's Embedded Vision Development Kit. It assumes that the reader is familiar with the basic Lattice FPGA design flow and mainly focuses on the Machine Learning part of the overall development process. For detailed instructions of the design flow described in this document, refer to the ECP5-Face-Identification-Reference-Design (FPGA-RD-02062).

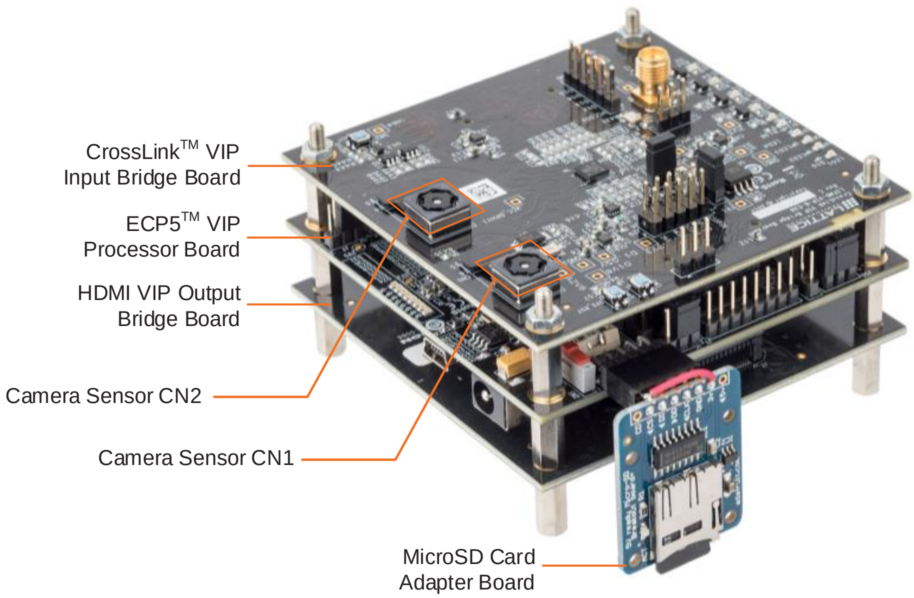


Figure 1.1. Lattice EVDK with MicroSD Card Adapter Board

1.1. Design Process Overview

The design process involves the following steps:

- Setting up the basic environment
- Preparing the dataset
- Training the machine
- Creating the frozen file (*.pb)
- Creating the binary file with Lattice sensAI™ 2.1 program
- Programming the binary and bitstream files to VIP board and SD card

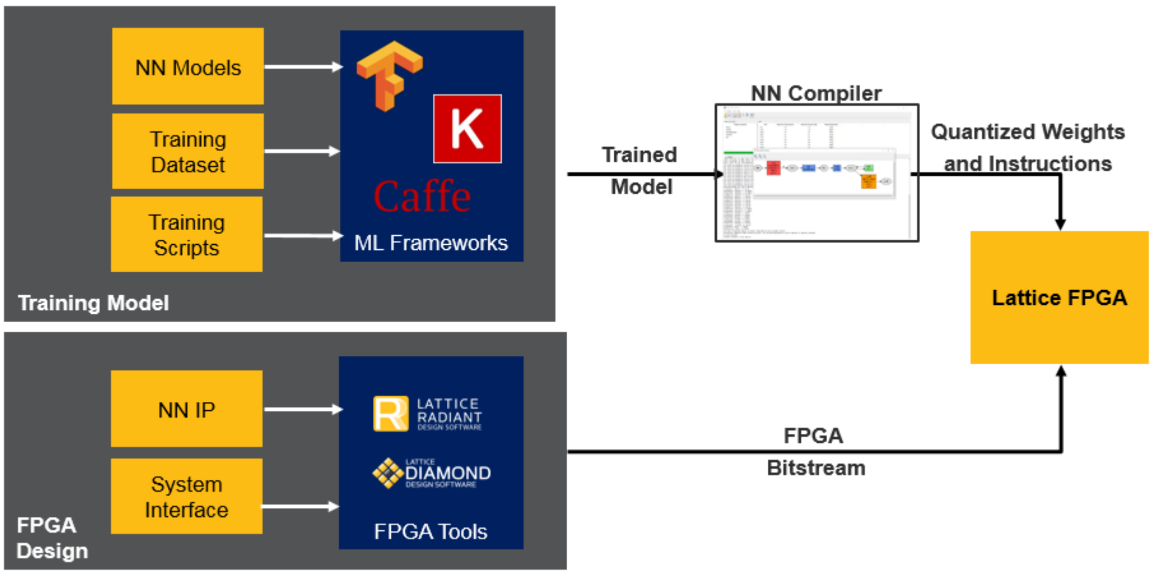


Figure 1.2. Lattice Machine Learning Design Flow

2. Machine Training and Creating Frozen file

2.1. Caffe Installation

For more information on how to install Caffe, go to https://caffe.berkeleyvision.org/.

Note: Caffe has some dependencies of environment and library packages, Lattice cannot help on resolving all installation issues caused by system dependencies.

2.2. Preparing the Dataset

Use the links below to download the training and testing VGGFace2 dataset:

- Training dataset – http://zeus.robots.ox.ac.uk/vgg_face2/get_file?fname=vggface2_train.tar.gz
- Testing dataset – http://zeus.robots.ox.ac.uk/vgg_face2/get_file?fname=vggface2_test.tar.gz

For the detailed procedure in preparing the dataset, refer to the Preparing the Dataset section in ECP5-Face-Identification-Reference-Design (FPGA-RD-02062).

2.3. Training the Machine with Caffe

1. Check three script files in the *train* folder as shown in Figure 2.1.

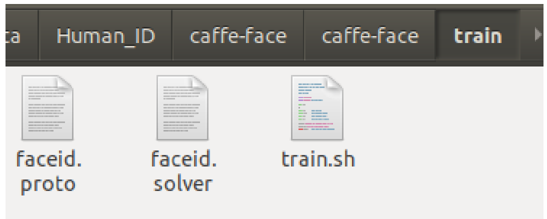


Figure 2.1. Default Content of the Train Folder

2. Copy the label files *train_new.txt* and *test_new.txt* to the *train* folder.

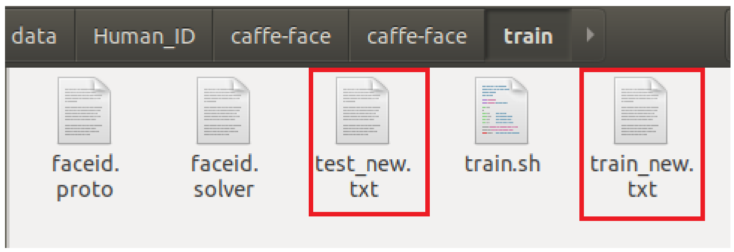


Figure 2.2. Machine Training Contents

3. Update the proto file *faceid.proto* for Image dataset source.

```
name: "FaceID_vgg8"
layer {
  name: "train_data"
  type: "ImageData"
  top: "data"
  top: "label"
  transform_param {
    mirror: true
    #crop_size: 90
    mean_value: 128
    mean_value: 128
    mean_value: 128
    max_rotation_angle: 20
  }
  image_data_param {
    source: "train_new.txt"
    #batch_size: 128
    #batch_size: 64
    batch_size: 32
    #batch_size: 16
    new_height: 90
    new_width: 90
    shuffle: true
  }
  include { phase: TRAIN }
}
layer {
  name: "test_data"
  type: "ImageData"
  top: "data"
  top: "label"
  transform_param {
    mirror: false
    mean_value: 128
    mean_value: 128
    mean_value: 128
  }
  image_data_param {
    source: "test_new.txt"
    batch_size: 10
```

Figure 2.3. Update Proto File for Dataset Label File Name

4. Set the output number of FC6 layer.

   Set '*num_output*' to the number of dataset class.

   The training code of face identification is for the object classification as MNIST application. Therefore, the final output number of the fully-connected model is the number of object class. The default value of FC6 output is 9295.

```
  top: "fc6"
  exclude: { phase: TEST not_stage: "val" }
  param {
    lr_mult: 1.0
    decay_mult: 1.0
  }
  param {
    lr_mult: 2.0
    decay_mult: 0.0
  }
  inner_product_param {
    num_output: 9295      The number of dataset class
    weight_filler {
      type: "xavier"
    }
    bias_filler {
      type: "constant"
      value: 0.1
    }
  }
}
```

Figure 2.4. Last Fully-connected Layer Output Number Setting

5. Update the following variables in train.sh as per your environment setup.
   - CAFFE_ROOT – with compiled caffe location.
   - OPT – GPU number or comment out to use CPU only.

```
Open ▾                      train.sh                          Save  ≡
            ~/data/Human_ID/caffe-face/caffe-face/train
################################################################################
#
################################################################################
SOLVER=faceid.solver
#WEIGHTS=fid_iter_100000.caffemodel
#SNAPSHOT=fid_iter_100000.solverstate

################################################################################
if [ -n "$SNAPSHOT" ]; then
  OPT=--snapshot=$SNAPSHOT
elif [ -n "$WEIGHTS" ]; then
  OPT=--weights=$WEIGHTS
fi
if [ "$CAFFE_ROOT" == "" ]; then
  #CAFFE_ROOT=$HOME/ML/caffe
fi
OPT="$OPT --gpu=0"
PYTHONPATH=$CAFFE_ROOT/python $CAFFE_ROOT/build/tools/caffe train --solver=$SOLVER $OPT |& tee -a
train.log
```

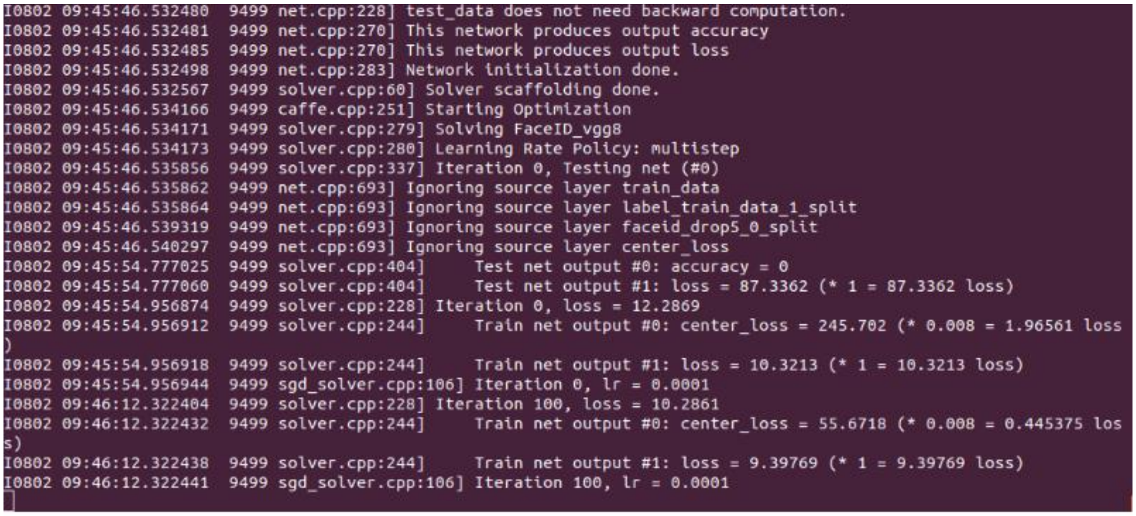
Figure 2.5. train.sh Parameter Setting Example

6. Run the *train.sh* script.

```
daniel@daniel:~/data/Human_ID/caffe-face/caffe-face/train$ ls
faceid.proto  faceid.solver  test_new.txt  train.log  train_new.txt  train.sh
daniel@daniel:~/data/Human_ID/caffe-face/caffe-face/train$ ./train.sh
```

Figure 2.6. Executing train.sh Script Command

Figure 2.7 shows a training status example.



```
I0802 09:45:46.532480  9499 net.cpp:228] test_data does not need backward computation.
I0802 09:45:46.532481  9499 net.cpp:270] This network produces output accuracy
I0802 09:45:46.532485  9499 net.cpp:270] This network produces output loss
I0802 09:45:46.532498  9499 net.cpp:283] Network initialization done.
I0802 09:45:46.532567  9499 solver.cpp:60] Solver scaffolding done.
I0802 09:45:46.534166  9499 caffe.cpp:251] Starting Optimization
I0802 09:45:46.534171  9499 solver.cpp:279] Solving FaceID_vgg8
I0802 09:45:46.534173  9499 solver.cpp:280] Learning Rate Policy: multistep
I0802 09:45:46.535856  9499 solver.cpp:337] Iteration 0, Testing net (#0)
I0802 09:45:46.535862  9499 net.cpp:693] Ignoring source layer train_data
I0802 09:45:46.535864  9499 net.cpp:693] Ignoring source layer label_train_data_1_split
I0802 09:45:46.539319  9499 net.cpp:693] Ignoring source layer faceid_drop5_0_split
I0802 09:45:46.540297  9499 net.cpp:693] Ignoring source layer center_loss
I0802 09:45:54.777025  9499 solver.cpp:404]     Test net output #0: accuracy = 0
I0802 09:45:54.777060  9499 solver.cpp:404]     Test net output #1: loss = 87.3362 (* 1 = 87.3362 loss)
I0802 09:45:54.956874  9499 solver.cpp:228] Iteration 0, loss = 12.2869
I0802 09:45:54.956912  9499 solver.cpp:244]     Train net output #0: center_loss = 245.702 (* 0.008 = 1.96561 loss
)
I0802 09:45:54.956918  9499 solver.cpp:244]     Train net output #1: loss = 10.3213 (* 1 = 10.3213 loss)
I0802 09:45:54.956944  9499 sgd_solver.cpp:106] Iteration 0, lr = 0.0001
I0802 09:46:12.322404  9499 solver.cpp:228] Iteration 100, loss = 10.2861
I0802 09:46:12.322432  9499 solver.cpp:244]     Train net output #0: center_loss = 55.6718 (* 0.008 = 0.445375 los
s)
I0802 09:46:12.322438  9499 solver.cpp:244]     Train net output #1: loss = 9.39769 (* 1 = 9.39769 loss)
I0802 09:46:12.322441  9499 sgd_solver.cpp:106] Iteration 100, lr = 0.0001
```

Figure 2.7. Training Status Example

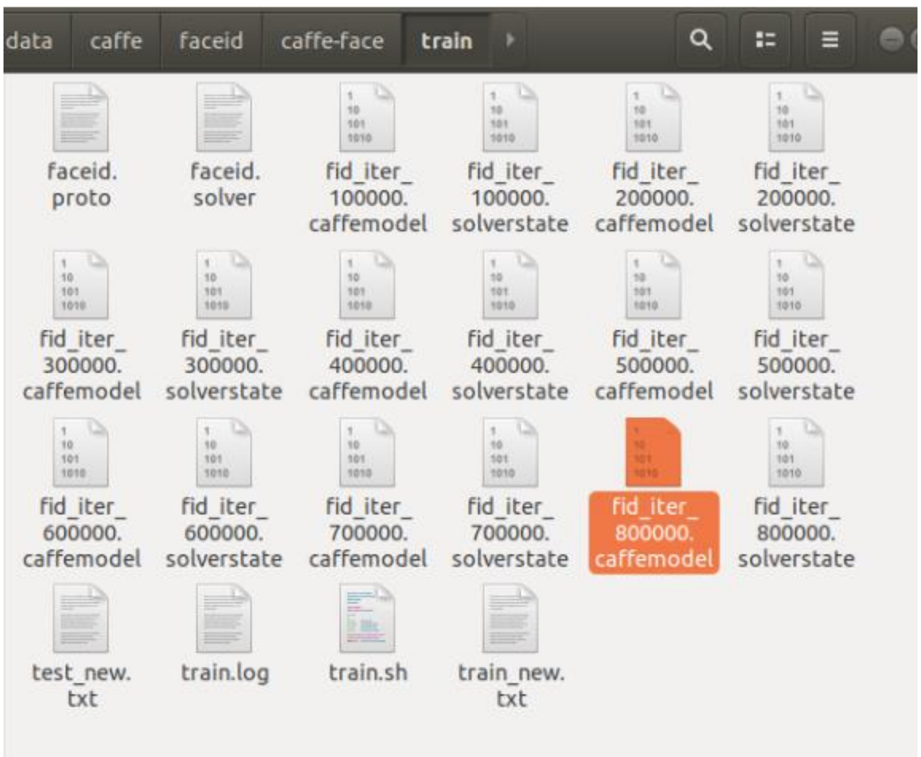


Figure 2.8. Machine Training Output File

7. Create the inference network model proto file.
   The inference network mode is different from the training proto file. The inference model does not use *Drop5* and *FC6* layer.
   Remove train_data, test_data, drop5, loss, accuracy, fc6, and center_loss.
   Lattice provides the reference proto file of Inference model file *faceid_demo.proto*.
   The inference proto file is used in the part of generating binary file.
   Refer to the Preparing the Dataset section in ECP5-Face-Identification-Reference-Design (FPGA-RD-02062).

2.4. Training the Machine with TensorFlow

To train the machine with TensorFlow:

1. Run the command in the *code root* directory:

```
$ python face-identification.py --data_dir=<path of data directory>
```

```
[earth]$ python3 face-identification.py --data_dir /home/earth/ssd/faceId/dataset/train_aug
Using TensorFlow backend.
Number of clases are  8741
name: GeForce RTX 2080 Ti major: 7 minor: 5 memoryClockRate(GHz): 1.755
pciBusID: 0000:01:00.0
totalMemory: 10.73GiB freeMemory: 4.33GiB
Found 5688962 images belonging to 8741 classes.
Found 628237 images belonging to 8741 classes.
Starting thread 139937718466304
Starting thread 139937710073600
Starting thread 139938246928128
Starting Training...!!!
Starting thread 139938255320832
2019-07-09 11:18:17: Step 0 softmax_loss: 9.1922, Center_loss: 98725.46(* 0.003=296.17638), ACU: 0.0
2019-07-09 11:18:37: Step 100 softmax_loss: 9.0761, Center_loss: 349.0126(* 0.003=1.04704), ACU: 0.0002
2019-07-09 11:18:57: Step 200 softmax_loss: 9.0754, Center_loss: 1.9106(* 0.003=0.00573), ACU: 0.0002
2019-07-09 11:19:19: Step 300 softmax_loss: 9.075, Center_loss: 0.0773(* 0.003=0.00023), ACU: 0.0
2019-07-09 11:19:44: Step 400 softmax_loss: 9.075, Center_loss: 0.026(* 0.003=8e-05), ACU: 0.0
2019-07-09 11:20:09: Step 500 softmax_loss: 9.0746, Center_loss: 0.0148(* 0.003=4e-05), ACU: 0.0002
2019-07-09 11:20:34: Step 600 softmax_loss: 9.0745, Center_loss: 0.0104(* 0.003=3e-05), ACU: 0.0
2019-07-09 11:20:58: Step 700 softmax_loss: 9.0745, Center_loss: 0.0071(* 0.003=2e-05), ACU: 0.0
2019-07-09 11:21:23: Step 800 softmax_loss: 9.0744, Center_loss: 0.0056(* 0.003=2e-05), ACU: 0.0002
2019-07-09 11:21:49: Step 900 softmax_loss: 9.0744, Center_loss: 0.0046(* 0.003=1e-05), ACU: 0.0
2019-07-09 11:22:14: Step 1000 softmax_loss: 9.0743, Center_loss: 0.0039(* 0.003=1e-05), ACU: 0.0
```

Figure 2.9. Machine Training Output File

2. Check the training status on TensorBoard by running the command below.

```
$ tensorboard –logdir=<path of log directory>
```

```
[earth@bangalore2 Final_code]$ tensorboard --logdir=logs
TensorBoard 1.14.0 at http://bangalore2.sibshivalik.com:6006/ (Press CTRL+C to quit)
I0702 11:07:43.642186 140085573707520 _internal.py:122] ::ffff:192.168.1.8 - - [02/Jul/2019 11:07:43] "GET / HTT
P/1.1" 200 -
I0702 11:07:45.654393 140085573707520 _internal.py:122] ::ffff:192.168.1.8 - - [02/Jul/2019 11:07:45] "GET /font
-roboto/oMMgfZMQthOryQo9n22dcuvvDin1pK8aKteLpeZ5c0A.woff2 HTTP/1.1" 200 -
```

Figure 2.10. TensorBoard – Launch

2.5. Generating the Frozen (*pb) File

Use the command below to generate the frozen protobuf(.pb) file:

```
$ python face-identification.py --log_dir=<path of log(checkpoint) directory> --freeze_model
```

```
[earth]$ python3 face-identification.py --log_dir /home/earth/ssd/faceId/tensorflow/Final_code/logs --freeze_mod
el
Face_Identification1000000.pbtxt saved at path /home/earth/ssd/faceId/tensorflow/Final_code/logs
/home/earth/ssd/faceId/tensorflow/Final_code/logs
inputShape shape [1, 90, 90, 3]
op: "Placeholder"
```

Figure 2.11. *pb File Generation from Checkpoint

3. Generating the Binary File

For the detailed procedure in creating the binary file, refer to the Creating Binary File with sensAI section ECP5-Face-Identification-Reference-Design (FPGA-RD-02062).

3.1. Programming the Bitstream and Binary files to VIP Board and SD Card

For the detailed procedure in flashing the bitstream file to the VIP board and flashing the binary file to the SD card, refer to refer to the Preparing the Dataset section in ECP5-Face-Identification-Reference-Design (FPGA-RD-02062).

Technical Support Assistance

Submit a technical support case through www.latticesemi.com/techsupport.

Revision History

Revision 1.0, October 2019

Section	Change Summary
All	Initial release.