Human Presence Detection Using Compact CNN Accelerator IP

Reference Design

FPGA-RD-02059-2.1

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

Acronyms in This Document 7
1. Introduction 8
1.1. Design Process Overview 8
2. Setting up the Basic Environment 9
2.1. Software and Hardware Requirements 9
2.1.1. Software 9
2.1.2. Hardware 9
2.2. Setting up the Linux Environment for Machine Training 10
2.2.1. Installing the NVIDIA CUDA and cuDNN Library for ML Training on GPU 10
2.2.2. Setting Up the Environment for Training and Model Freezing Scripts 11
2.2.3. Installing TensorFlow v1.12 13
2.2.4. Installing the Python Package 14
3. Preparing the Dataset 16
3.1. Downloading the Dataset 16
3.2. Visualizing and Tuning/Cleaning Up the Dataset 18
3.3. Data Augmentation 19
3.3.1. Configuring the Augmentation 19
3.3.2. Running the Augmentation 20
4. Training the Machine 21
4.1. Training Code Structure 21
4.2. Neural Network Architecture 22
4.2.1. Neural Network Architecture 22
4.2.2. Human Presence Detection Network Output 25
4.2.3. Training Code Overview 26
4.3. Training from Scratch and/or Transfer Learning 33
5. Creating Frozen File 37
5.1. Generating the frozen .pb File 37
6. Creating Binary File with SensAI 38
7. Hardware (RTL) Implementation 42
7.1. Top Level Information 42
7.1.1. Block Diagram 42
7.1.2. Overall Operational Flow 42
7.1.3. Core Customization 43
7.2. Architectural Details 43
7.2.1. CNN Pre-Processing 43
7.2.2. CNN Post-Processing (humandet_post.v) 46
8. Creating FPGA Bitstream File 49
9. Running the iCE40 Human Presence Detection Demo 52
9.1. Functional Description 52
9.2. Programming Human Presence Detection Demo on iCE40 SPI Flash 52
9.3. Running iCE40 Human Presence Detection Demo on Hardware 57
Appendix A. Other Labelling Tools 58
References 59
Technical Support Assistance 60
Revision History 61

Figures

Figure 1.1. Lattice Machine Learning Design Flow 8
Figure 2.1. HiMax HM01B0 UPduino Shield Board 9
Figure 2.2. CUDA Repo Download 10
Figure 2.3. CUDA Repo Installation 10
Figure 2.4. Fetch Keys 10
Figure 2.5. Updated Ubuntu Package Repositories 10
Figure 2.6. CUDA Installation Completed 11
Figure 2.7. cuDNN Installation 11
Figure 2.8. Anaconda Package Download 11
Figure 2.9. Anaconda Installation 12
Figure 2.10. License Terms Prompt 12
Figure 2.11. Installation Path Confirmation 12
Figure 2.12. Launch/Initialize Anaconda Environment on Installation Completed 12
Figure 2.13.Anaconda Environment Activation 13
Figure 2.14. TensorFlow Installation 13
Figure 2.15. TensorFlow Installation Confirmation 13
Figure 2.16. TensorFlow Installation Completed 13
Figure 2.17. Easydict Installation 14
Figure 2.18. Joblib Installation 14
Figure 2.19. Keras Installation 14
Figure 2.20. OpenCV Installation 14
Figure 2.21. Pillow Installation 15
Figure 3.1. Open Source Dataset Repository Cloning 16
Figure 3.2. OIDv4_Toolkit Directory Structure 16
Figure 3.3. Dataset Script Option/Help 17
Figure 3.4. Dataset Downloading Logs 17
Figure 3.5. Downloaded Dataset Directory Structure 17
Figure 3.6. OIDv4 Label to KITTI Format Conversion 17
Figure 3.7. Toolkit Visualizer 18
Figure 3.8. Manual Annotation Tool – Cloning 18
Figure 3.9. Manual Annotation Tool – Directory Structure 18
Figure 3.10. Manual Annotation Tool – Launch 19
Figure 3.11. Augmentation Directory Stucture 19
Figure 3.12. config.py Configuration File Parameters 19
Figure 3.13. Selecting the Augmentation Operations 20
Figure 3.14. Running the Augmentataion 20
Figure 4.1. Training Code Directory Structure 21
Figure 4.2. Model Layer Dimensions 23
Figure 4.3. Model Output Format 25
Figure 4.4. Training Code Flow Diagram 26
Figure 4.5. Code Snippet – Input Image Size Config 27
Figure 4.6. Code Snippet – Input Image Size Config (Grid Sizes) 27
Figure 4.7. Code Snippet – Batch Image Size Config 27
Figure 4.8. Code Snippet – Anchors per Grid Config #1 27
Figure 4.9. Code Snippet – Anchors per Grid Config #2 28
Figure 4.10. Code Snippet – Anchors per Grid Config #3 28
Figure 4.11. Code Snippet – Training Parameters 28
Figure 4.12. Code Snippet – Quantization Value Setting 29
Figure 4.13. Code Snippet – Forward Graph Fire Layers 29
Figure 4.14. Code Snippet – Forward Graph Last Convolution Layer 30
Figure 4.15. Code Snippet – Quantization Layer 30
Figure 4.16. Code Snippet – Interpret Output Graph 31

Figure 4.17. Code Snippet – Class Loss 32
Figure 4.18. Code Snippet – Bbox Loss 32
Figure 4.19. Code Snippet – Confidence Loss 33
Figure 4.20. Training Code Snippet for Mean and Scale 33
Figure 4.21. Training Code Snippet for Dataset Path 33
Figure 4.22. Create File for Dataset train.txt 34
Figure 4.23. Training Input Parameter..34
Figure 4.24. Execute Run Script 34
Figure 4.25. TensorBoard – Generated Link 35
Figure 4.26. TensorBoard 35
Figure 4.27. Image Menu of TensorBoard 35
Figure 4.28. Example of Checkpoint Data Files at Log Folder 36
Figure 5.1. pb File Generation from Checkpoint 37
Figure 5.2. Frozen pb File..37
Figure 6.1. SensAI Home Screen 38
Figure 6.2. SensAI –Network File Selection 39
Figure 6.3. SensAI –Image Data File Selection 39
Figure 6.4. SensAI – Project Settings 40
Figure 6.5. SensAI – Analyze Project...40
Figure 7.1. Top Level Block Diagram Human Presence Detection iCE40 42
Figure 7.2. Image Zoning Enabled...43
Figure 7.3. RTL logic – Zone Counter 44
Figure 7.4. Masking for Zone 1 44
Figure 7.5. Downscaling Zones 1-5 45
Figure 7.6. Downscaling Zone 6 45
Figure 7.7. Image Zoning Disabled..46
Figure 7.8. RTL Logic – Maximum CNN Value Calculation 47
Figure 7.9. RTL Logic – Driving Output LED Logic[1] 47
Figure 7.10. RTL Logic – Driving Output LED Logic[2] 47
Figure 8.1. Radiant Software 49
Figure 8.2. Radiant Software – Open Project 50
Figure 8.3. Radiant Software – Bitstream Generation..50
Figure 8.4. Radiant Software – Bitstream Generation Export Report 51
Figure 9.1. iCE40 Human Presence Demo Diagram...52
Figure 9.2. Radiant Programmer – Creating New Project 53
Figure 9.3. Radiant Programmer – iCE40 UltraPlus Device Family Selection 53
Figure 9.4. Radiant Programmer – iCE40 UltraPlus Device Selection..54
Figure 9.5. Radiant Programmer – Bitstream Flashing Settings 55
Figure 9.6. Radiant Programmer – Firmware Bin File Flashing Setting 56
Figure 9.7. Camera and LED Location 57

Tables

Table 4.1. Convolution Network Configuration of Human Presence Detection Design 22
Table 7.1. Core Parameters 43
Table A.1. Other Labelling Tools 58

Acronyms in This Document

A list of acronyms used in this document.

Acronym	Definition
CKPT	Checkpoint
CNN	Convolutional Neural Network
cuDNN	CUDA® Deep Neural Network
EVDK	Embedded Vision Development Kit
FPGA	Field-Programmable Gate Array
GPU	Graphics Processing Unit
LED	Light-emitting diode
ML	Machine Learning
MLE	Machine Learning Engine
NN	Neural Network
NNC	Neural Network Compiler
SD	Secure Digital
SDHC	Secure Digital High Capacity
SDXC	Secure Digital eXtended Capacity
SPI	Serial Peripheral Interface
USB	Universal Serial Bus
VIP	Video Interface Platform

1. Introduction

This document describes the Human Presence Detection design process using an iCE40 UltraPlus™ FPGA platform (HiMax HM01B0 UPduino Shield).

1.1. Design Process Overview

The design process involves the following steps:

1. Training the model
    - Setting up the basic environment
    - Preparing the dataset.
        - Preparing the 64 x 64 image
        - Labeling dataset of human bounding box
    - Training the machine
        - Training the machine and creating the checkpoint data
    - Creating the frozen file (*.pb)
2. Compiling Neural Network
    - Creating the binary file with Lattice SensAI 2.1 program
3. FPGA design
    - Creating the FPGA Bitstream file
4. FPGA Bitstream and Quantized Weights and Instructions
    - Flashing the binary and bitstream files to iCE40 UPduino hardware

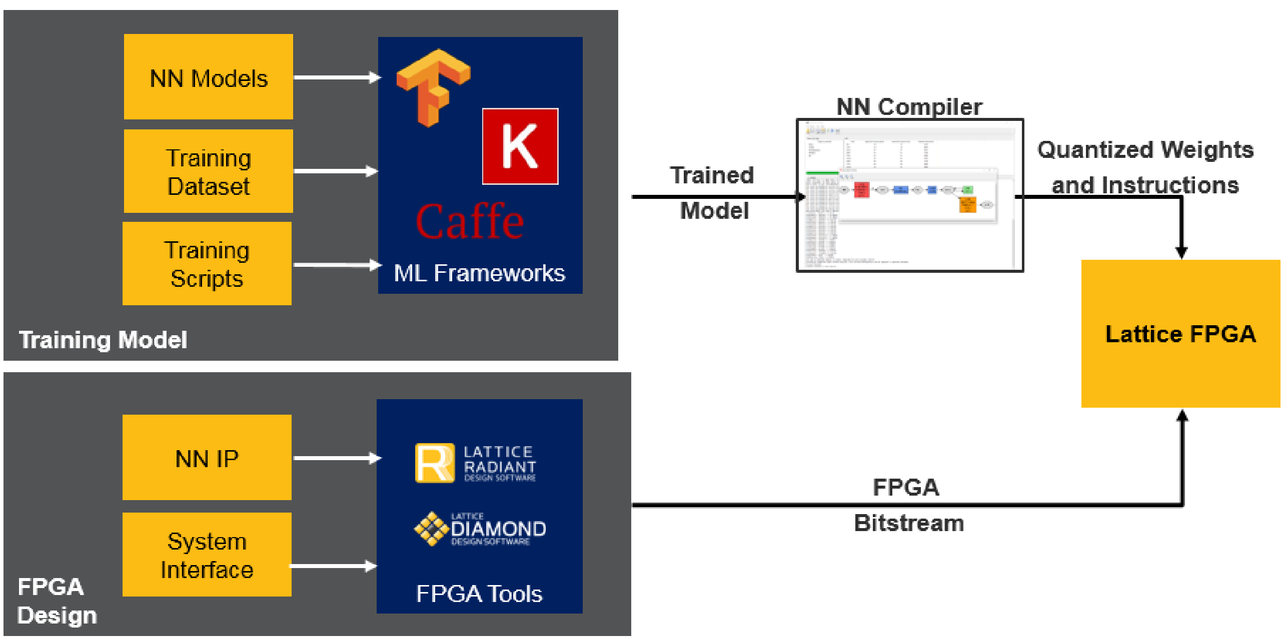

Figure 1.1. Lattice Machine Learning Design Flow

2. Setting up the Basic Environment

2.1. Software and Hardware Requirements

This section describes the required tools and environment setup for training and model freezing.

2.1.1. Software

- Lattice Radiant Software
  Refer to http://www.latticesemi.com/latticeradiant
- Lattice Radiant Programmer
  Refer to http://www.latticesemi.com/programmer
- Neural Network Compiler version 2.1
  Refer to https://www.latticesemi.com/Products/DesignSoftwareAndIP/AIML/NeuralNetworkCompiler

2.1.2. Hardware

This design uses the HiMax HM01B0 UPduino Shield as shown in Figure 2.1.
Refer to http://www.latticesemi.com/en/Products/DevelopmenBoardsAndKits/HimaxHM01B0.

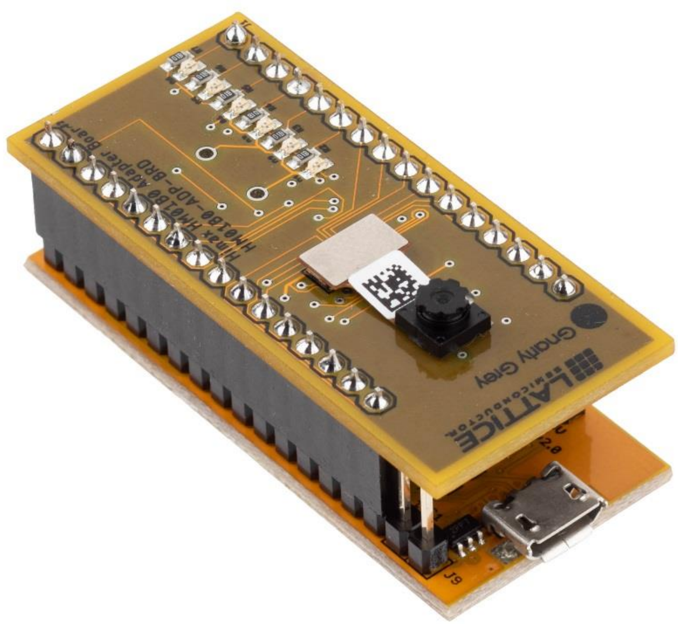

Figure 2.1. HiMax HM01B0 UPduino Shield Board

2.2. Setting up the Linux Environment for Machine Training

This section describes the steps for NVIDIA GPU drivers and/or libraries for 64-bit **Ubuntu 16.04 OS**.

Note: NVIDIA library and TensorFlow version is dependent on PC and Ubuntu/Windows version.

2.2.1. Installing the NVIDIA CUDA and cuDNN Library for ML Training on GPU

2.2.1.1. Installing the CUDA Toolkit

To install the NVIDIA CUDA toolkit, run the commands below:

1. Download the NVIDIA CUDA toolkit.

```
$ curl -O
https://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/cu
da-repo-ubuntu1604_10.1.105-1_amd64.deb
```

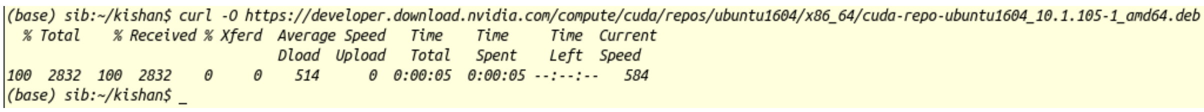

Figure 2.2. CUDA Repo Download

2. Install the deb package.

```
$ sudo dpkg -i ./cuda-repo-ubuntu1604_10.1.105-1_amd64.deb
```

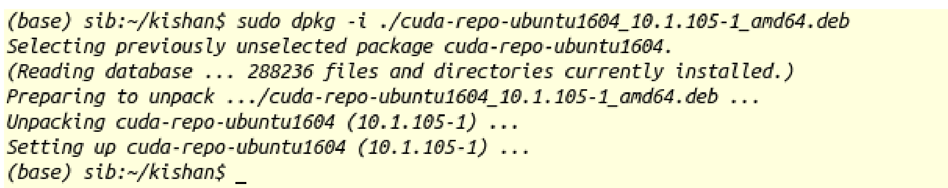

Figure 2.3. CUDA Repo Installation

3. Proceed with the installation.

```
$ sudo apt-key adv --fetch-keys
http://developer.download.nvidia.com/compute/cuda/repos/ubuntu1604/x86_64/7fa
2af80.pub
```

Figure 2.4. Fetch Keys

```
$ sudo apt-get update
```

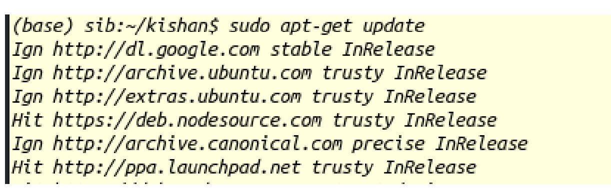

Figure 2.5. Updated Ubuntu Package Repositories

```
sudo apt-get install cuda-9-0
```

```
(base) sib:~/kishan$ sudo apt-get install cuda-9-0
Reading package lists... Done
Building dependency tree
Reading state information... Done
```

Figure 2.6. CUDA Installation Completed

2.2.1.2. Installing the cuDNN

To install cuDNN:

1. Create your NVIDIA developer account in https://developer.nvidia.com.
2. Download the cuDNN library from https://developer.nvidia.com/compute/machine-learning/cudnn/secure/v7.1.4/prod/9.0_20180516/cudnn-9.0-linux-x64-v7.1.
3. Run the commands below to install cuDNN:

```
$ tar xvf cudnn-9.0-linux-x64-v7.1.tgz
$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
$ sudo chmod a+r /usr/local/cuda/include/cudnn.h
/usr/local/cuda/lib64/libcudnn*
```

```
k$ tar xvf cudnn-9.0-linux-x64-v7.1.tgz
cuda/include/cudnn.h
cuda/NVIDIA_SLA_cuDNN_Support.txt
cuda/lib64/libcudnn.so
cuda/lib64/libcudnn.so.7
cuda/lib64/libcudnn.so.7.1.4
cuda/lib64/libcudnn_static.a
k$ sudo cp cuda/include/cudnn.h /usr/local/cuda/include
k$ sudo cp cuda/lib64/libcudnn* /usr/local/cuda/lib64
k$ sudo chmod a+r /usr/local/cuda/include/cudnn.h /usr/local/cuda/lib64/libcudnn*
k$ _
```

Figure 2.7. cuDNN Installation

2.2.2. Setting Up the Environment for Training and Model Freezing Scripts

This section describes the environment setup for training and model freezing scripts for 64-bit Ubuntu 16.04.

Anaconda provides one of the easiest ways to perform machine learning development and training on Linux.

2.2.2.1. Installing the Anaconda and Python3

To install the Anaconda and Python 3:

1. Go to https://www.anaconda.com/distribution/#download.
2. Download Python 3 version of Anaconda for Linux.

```
sib:~/kishan$ wget https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
--2019-04-18 11:34:54--  https://repo.anaconda.com/archive/Anaconda3-2019.03-Linux-x86_64.sh
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.131.3, 104.16.130.3, 2606:4700::6810:8303, ...
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.131.3|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 685906562 (654M) [application/x-sh]
Saving to: 'Anaconda3-2019.03-Linux-x86_64.sh'

100%[==================================================================================================
```

Figure 2.8. Anaconda Package Download

3. Run the command below to install the Anaconda environment.

```
$ sh Anaconda3-2019.03-Linux-x86_64.sh
```

Note: Anaconda3-<version>-Linux-x86_64.sh version may vary based on the release.

```
sib:~/kishan$ sh Anaconda3-2019.03-Linux-x86_64.sh

Welcome to Anaconda3 2019.03

In order to continue the installation process, please review the license
agreement.
Please, press ENTER to continue
>>> _
```

Figure 2.9. Anaconda Installation

4. Accept the license.

```
Do you accept the license terms? [yes|no]
[no] >>> yes_
```

Figure 2.10. License Terms Prompt

5. Confirm the installation path. Follow the instructions onscreen to change the default path.

```
Do you accept the license terms? [yes|no]
[no] >>> yes

Anaconda3 will now be installed into this location:
/home/sibridge/anaconda3

  - Press ENTER to confirm the location
  - Press CTRL-C to abort the installation
  - Or specify a different location below

[/home/sibridge/anaconda3] >>> /home/sibridge/kishan/anaconda3_
```

Figure 2.11. Installation Path Confirmation

6. After installation, enter **No** as shown in Figure 2.12.

```
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> no_
```

Figure 2.12. Launch/Initialize Anaconda Environment on Installation Completed

2.2.3. Installing TensorFlow v1.12

To install the TensorFlow v1.12:

1. Activate the conda environment by running the command below:

```
$ source <conda  directory>/bin/activate
```

```
sib:~/kishan$ source anaconda3/bin/activate
(base) sib:~/kishan$ _
```

Figure 2.13.Anaconda Environment Activation

2. Install the TensorFlow by running the command below:

```
$ conda install tensorflow-gpu==1.12.0
```

```
(base) sib:~/kishan$ conda install tensorflow-gpu==1.12.0
WARNING: The conda.compat module is deprecated and will be removed in a future release.
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - tensorflow-gpu==1.12.0
```

Figure 2.14. TensorFlow Installation

3. After installation, enter Y as shown in Figure 2.15.

```
  wurlitzer                              1.0.2-py37_0 --> 1.0.2-py36_0
  xlrd                                   1.2.0-py37_0 --> 1.2.0-py36_0
  xlwt                                   1.3.0-py37_0 --> 1.3.0-py36_0
  zict                                   0.1.4-py37_0 --> 0.1.4-py36_0
  zipp                                   0.3.3-py37_1 --> 0.3.3-py36_1


Proceed ([y]/n)? y_
```

Figure 2.15. TensorFlow Installation Confirmation

Figure 2.16 shows TensorFlow installation is completed.

```
Preparing transaction: done
Verifying transaction: done
Executing transaction: done
(base) sib:~/kishan$ _
```

Figure 2.16. TensorFlow Installation Completed

2.2.4. Installing the Python Package

To install the Python package:

1. Install Easydict by running the command below.

```
$ conda install -c conda-forge easydict
```

```
(base) sib:~/kishan$ conda install -c conda-forge easydict
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - easydict
```

Figure 2.17. Easydict Installation

2. Install joblib by running the command below.

```
$ conda install joblib
```

```
(base) sib:~/kishan$ conda install joblib
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - joblib
```

Figure 2.18. Joblib Installation

3. Install Keras by running the command below.

```
$ conda install keras
```

```
(base) sib:~/kishan$ conda install joblib
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - joblib
```

Figure 2.19. Keras Installation

4. Install OpenCV by running the command below.

```
$ conda install opencv
```

```
(base) sib:~/kishan$ conda install opencv
Collecting package metadata: done
Solving environment: done

## Package Plan ##

  environment location: /home/sibridge/kishan/anaconda3

  added / updated specs:
    - opencv
```

Figure 2.20. OpenCV Installation

5. Install Pillow by running the command below.

```
$ conda install pillow
```

```
(base) sib:~/kishan$ conda install pillow
Collecting package metadata: done
Solving environment: done

# All requested packages already installed.

(base) sib:~/kishan$ _
```

Figure 2.21. Pillow Installation

3. Preparing the Dataset

This chapter describes how to create a dataset using examples from Google Open Image Dataset.

The Google Open Image Dataset version 4 (https://storage.googleapis.com/openimages/web/index.html) features more than 600 classes of images. The Person class of images include human annotated and machine annotated labels and bounding box. Annotations are licensed by Google Inc. under CC BY 4.0 and images are licensed under CC BY 2.0.

3.1. Downloading the Dataset

To download the dataset, run the commands below.

1. Clone the OIDv4_Toolkit repository.

```
$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
$ cd OIDv4_ToolKit
```

```
(base) k$ git clone https://github.com/EscVM/OIDv4_ToolKit.git
Cloning into 'OIDv4_ToolKit'...
remote: Enumerating objects: 25, done.
remote: Counting objects: 100% (25/25), done.
remote: Compressing objects: 100% (24/24), done.
remote: Total 382 (delta 3), reused 14 (delta 1), pack-reused 357
Receiving objects: 100% (382/382), 34.06 MiB | 752.00 KiB/s, done.
Resolving deltas: 100% (111/111), done.
(base) k$
```

Figure 3.1. Open Source Dataset Repository Cloning

Figure 3.2 shows the OIDv4 directory structure.

```
OIDv4_ToolKit/
├── classes.txt
├── images
│   ├── classes.png
│   ├── rectangle.png
│   └── visualizer_example.gif
├── LICENSE
├── main.py
├── modules
│   ├── bounding_boxes.py
│   ├── csv_downloader.py
│   ├── downloader.py
│   ├── image_level.py
│   ├── parser.py
│   ├── show.py
│   ├── test.py
│   └── utils.py
├── README.md
└── requirements.txt
```

Figure 3.2. OIDv4_Toolkit Directory Structure

View the OIDv4 Toolkit Help menu.

```
$ python3 main.py -h
```

```
(base) k$ python3 main.py -h
usage: main.py [-h] [--Dataset /path/to/OID/csv/]
               [--classes list of classes [list of classes ...]]
               [--type_csv 'train' or 'validation' or 'test' or 'all']
               [--sub Subset of human verified images or machine generated h or m)]
               [--image_IsOccluded 1 or 0] [--image_IsTruncated 1 or 0]
               [--image_IsGroupOf 1 or 0] [--image_IsDepiction 1 or 0]
               [--image_IsInside 1 or 0] [--multiclasses 0 (default or 1]
               [--n_threads [default 20]] [--noLabels]
               [--limit integer number]
               <command> 'downloader', 'visualizer' or 'ill_downloader'.

Open Image Dataset Downloader

positional arguments:
  <command> 'downloader', 'visualizer' or 'ill_downloader'.
                        'downloader', 'visualizer' or 'ill_downloader'.
```

Figure 3.3. Dataset Script Option/Help

2. Use the OIDv4 Toolkit to download dataset. Download Person class images.

```
$ python3 main.py downloader --classes Person --type_csv validation
```

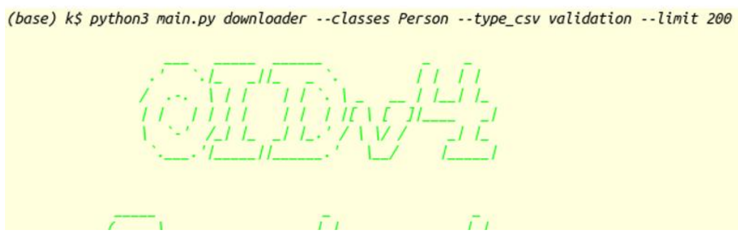

Figure 3.4. Dataset Downloading Logs

Figure 3.5 shows the downloaded dataset directory structure.

```
├── OID
│   ├── csv_folder
│   │   ├── class-descriptions-boxable.csv
│   │   └── validation-annotations-bbox.csv
│   └── Dataset
│       └── validation
│           └── Person
│               ├── ff7b4cc8ca9b6592.jpg
│               └── Label
│                   └── ff7b4cc8ca9b6592.txt
```

Figure 3.5. Downloaded Dataset Directory Structure

3. Lattice training code uses KITTI (.txt) format. The downloaded dataset is not in the required KITTI format. Convert the annotation to KITTI format.

```
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/train/Person/Label/*
$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/test/Person/Label/*
```

```
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 324.614144 69.905733 814.569472 681.9072
(base) k$ sed -i -- 's/Person/Person 0 0 0/g' OID/Dataset/validation/Person/Label/*
(base) k$ cat OID/Dataset/validation/Person/Label/ff7b4cc8ca9b6592.txt
Person 0 0 0 324.614144 69.905733 814.569472 681.9072
(base) k$
```

Figure 3.6. OIDv4 Label to KITTI Format Conversion

Note: KITTI Format: Person 0 0 0 324.61 69.90 814.56 681.90**.** It has class ID followed by truncated, occluded, alpha, Xmin, Ymin, Xmax, Ymax. The code converts Xmin, Ymin, Xmax, Ymax into x, y, w, h while training as *bounding box rectangle coordinates*.

3.2. Visualizing and Tuning/Cleaning Up the Dataset

To visualize and annotate the dataset, run the commands below:

1. Visualize the labelled images.

```
$ python3 main.py visualizer
```

Figure 3.7. Toolkit Visualizer

2. Clone the manual annotation tool from the GitHub repository.

```
$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
```

```
(base) k$ git clone https://github.com/SaiPrajwal95/annotate-to-KITTI.git
Cloning into 'annotate-to-KITTI'...
remote: Enumerating objects: 27, done.
remote: Total 27 (delta 0), reused 0 (delta 0), pack-reused 27
Unpacking objects: 100% (27/27), done.
(base) k$ _
```

Figure 3.8. Manual Annotation Tool – Cloning

3. Go to annotate-to-KITTI.

```
$ cd annotate-to-KITTI
$ ls
```

```
annotate-to-KITTI/
├── annotate-folder.py
└── README.md
```

Figure 3.9. Manual Annotation Tool – Directory Structure

4. Install the dependencies (OpenCV 2.4).

```
$ sudo apt-get install python-opencv
```

5. Launch the utility

```
$ python3 annotate-folder.py
```

6. Set the dataset path and default object label.

```
(base) k$ python3 annotate-folder.py
Enter the path to dataset: /tmp/images
Enter default object label: Person
[{'label': 'Person', 'bbox': {'xmin': 443, 'ymin': 48, 'xmax': 811, 'ymax': 683}}]
(base) k$ _
```

Figure 3.10. Manual Annotation Tool – Launch

7. For annotation, run the script provided in the website below.
   https://github.com/SaiPrajwal95/annotate-to-KITTI
   For more information on other labelling tools, see Appendix A. Other Labelling Tools.

3.3. Data Augmentation

Data Augmentation needs large amount of training data to achieve good performance. Image Augmentation creates training images through different ways of processing or combination of multiple processing such as random rotation, shifts, shear and flips, and so on.

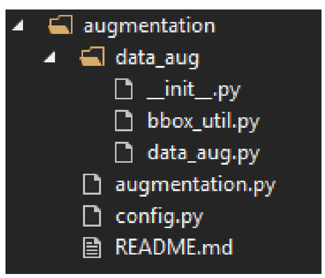

Figure 3.11. Augmentation Directory Stucture

- data_aug – It contains basic methods and augmentation classes.
- augmentation.py – This file reads the input images (input labels) and performs preferred augmentation on it.
- config.py – Contains parameters that are used in augmentation operations.

3.3.1. Configuring the Augmentation

To configure the augmentation:

1. Configure the *config.py* file which contains the parameters shown in Figure 3.12.

```
Input_dict = {
    'AngleForRotation': '90,190,270',
    'GammaForRandomBrightness1': 0.6,
    'GammaForRandomBrightness2': 1.5,
    'FilterSizeForGaussianFiltering': 11,
    'SnowCoeffForAddSnow': 0.5,
    'resizeheight': 64,
    'resizewidth': 64,
    }
```

Figure 3.12. config.py Configuration File Parameters

2. Choose the operations to perform on the dataset. The operations can be selected in *augmentation.py* by editing the list *all_op*.

```
all_op = [
    'RandomHorizontalFlip',
    #'RandomScale',
    #'RandomRotate',
    #'RandomTranslate',
    #'RandomShear',
    #'Rotate',
    'Translate',
    #'Shear',
    #'GaussianFiltering',
    'RandomBrightness2_0',
    'RandomBrightness0_5',
    #'Resize'
]
```

Figure 3.13. Selecting the Augmentation Operations

3. Add or Remove the operation by commenting/uncommenting the operation in the *all_op* list as shown in Figure 3.13.

3.3.2. Running the Augmentation

Run the augmentation by running the command below:

```
python augmentation.py --image_dir <Path_To_InputImage_Dir> --label_dir
<Path_To_InputLabel_Dir> --out_image_dir <Path_To_OutputImage_Dir> --
out_label_dir <Path_To_OutputLable_Dir>
```

```
earth:~$ python augmentation.py --image_dir /data/images/ --label_dir /data/labels/ --out_image_dir /tmp/labels --out_label_dir /tmp/images/

| ------------------------------------------- |
| Initial Configuration
| ------------------------------------------- |
| AngleForRotation                     | 90,190,270 |
| GammaForRandomBrightness1            | 0.6 |
| GammaForRandomBrightness2            | 1.5 |
| FilterSizeForGaussianFiltering       | 11  |
| SnowCoeffForAddSnow                  | 0.5 |
| resizeheight                         | 64  |
| resizewidth                          | 64  |
| ------------------------------------------|

Copying original images and labels ...
Done Coping

Running Augmentation...!!!
operation : RandomHorizontalFlip

Operation number  1 out of 4
ignoring image due to small bbox size for /data/images/a5a886c21a52344f.jpg
```

Figure 3.14. Running the Augmentataion

4. Training the Machine

4.1. Training Code Structure

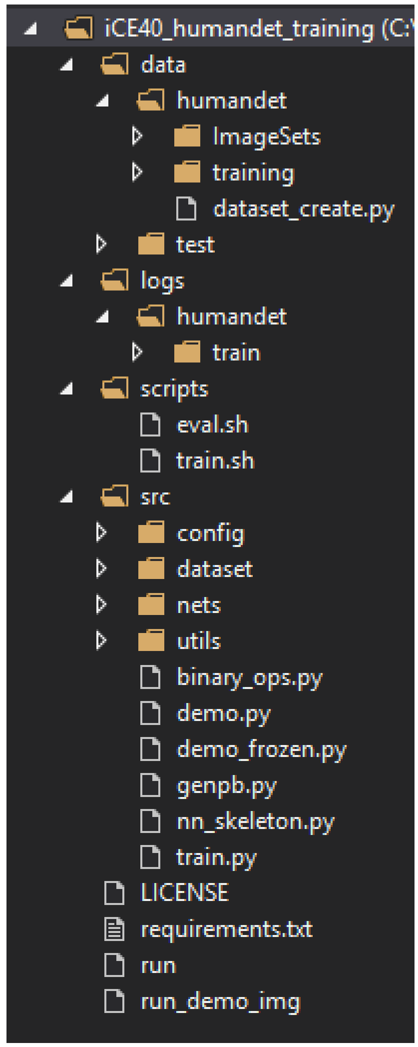

Figure 4.1. Training Code Directory Structure

4.2. Neural Network Architecture

4.2.1. Neural Network Architecture

This section provides information on the Convolution Network Configuration of the Human Presence Detection design. The Neural Network model of the Human Presence Detection design uses VGG NN base model and the detection layer of SqueezeDet model.

Table 4.1. Convolution Network Configuration of Human Presence Detection Design

Image Input (64 x 64 x 3)		
Fire 1	Conv3 – 16	Conv3 - # where: • *Conv3* – 3 x 3 Convolution filter Kernel size • *#* - The number of filter For example, Conv3 - 16 = 16 3 x 3 convolution filter BN – Batch Normalization FC - # where: • *FC* – Fully connected layer • *#* - The number of output
	BN	
	Relu	
	Maxpool	
Fire 2	Conv3 – 16	
	BN	
	Relu	
Fire 3	Conv3 – 32	
	BN	
	Relu	
	Maxpool	
Fire 4	Conv3 – 32	
	BN	
	Relu	
Fire 5	Conv3 – 32	
	BN	
	Relu	
	Maxpool	
Fire 6	Conv3 – 44	
	BN	
	Relu	
Fire 7	Conv3 – 48	
	BN	
	Relu	
	Maxpool	
Conv12	Conv3 – 42	

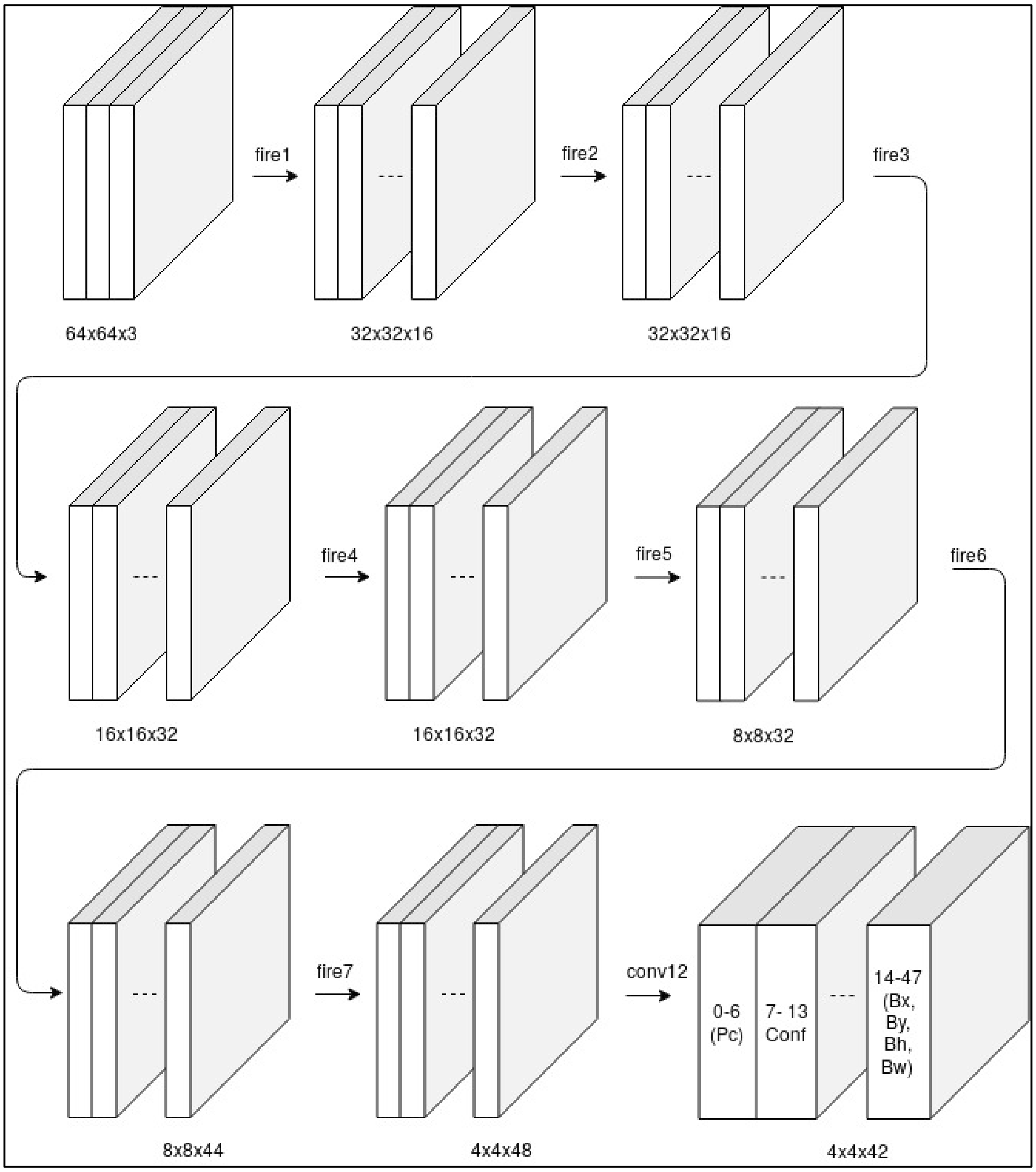

Figure 4.2. Model Layer Dimensions

- The Human Detection network structure consists of seven fire layers followed by one convolution layer. Fire layer contains convolution, batch normalization, and relu layers. Fire 1, Fire 3, Fire 5, and Fire 7 layers contain pooling, while Fire 2, Fire 4, and Fire 6 layers do not contain pooling.
- In Table 4.1, the layer contains convolution (conv), batch normalization (bn), and relu layers.
- Figure 4.2 shows the dimensions of each layer of the network.
- Layer information:
  - Convolutional Layer

    In general, the first layer in a CNN is always a convolutional layer**.** Each layer consists of number of filters (sometimes referred as kernels) which convolves with input layer/image and generates activation map (that is feature map). This filter is an array of numbers (the numbers are called weights or parameters). Each of these filters can be thought of as feature identifiers, like straight edges, simple colors, and curves and other high-level features. For example, the filters on the first layer convolve around the input image and *activate* (or compute high values) when the specific feature (for example,curve) it is looking for is in the input volume.

- Relu (Activation Layer)

  After each conv layer, it is convention to apply a nonlinear layer (or activation layer) immediately afterward. The purpose of this layer is to introduce nonlinearity to a system that basically has just been computing linear operations during the conv layers (element-wise, multiplications, and summations). In the past, nonlinear functions like tanh and sigmoid were used, but researchers found out that ReLU layers work far better because the network is able to train a lot faster (because of the computational efficiency) without making a significant difference to the accuracy. The ReLU layer applies the function f(x) = max (0, x) to all of the values in the input volume. In basic terms, this layer just changes all the negative activations to 0. This layer increases the nonlinear properties of the model and the overall network without affecting the receptive fields of the conv layer.

- Pooling Layer

  After some ReLu layers, you may choose to apply a pooling layer. It is also referred to as a down sampling layer. In this category, there are also several layer options, with max pooling being the most popular. This basically takes a filter (normally by size 2 x 2) and a stride of the same length. It then applies to the input volume and outputs the maximum number in every sub region that the filter convolves around.

  The intuitive reasoning behind this layer is that once you know that a specific feature is in the original input volume (there is a high activation value), its exact location is not as important as its relative location to the other features. This layer drastically reduces the spatial dimension (the length and the width change but not the depth) of the input volume. This serves two main purposes. First is that the number of parameters or weights is reduced by 75%, thus lessening the computation cost. Second is that it controls over fitting. This term refers to when a model is so tuned to the training examples that it is not able to generalize well for the validation and test sets. A symptom of over fitting is having a model that gets 100% or 99% on the training set, but only 50% on the test data.

- Batch Normalization

  Batch Normalization layer reduces the internal covariance shift. In order to train a neural network, perform preprocessing to the input data. For example, you can normalize all data so that it resembles a normal distribution (that means, zero mean and a unitary variance). This is to prevent the early saturation of non-linear activation functions like the sigmoid function, assuring that all input data is in the same range of values, and so on.

  But the problem appears in the intermediate layers because the distribution of the activations is constantly changing during training. This slows down the training process because each layer must learn to adapt themselves to a new distribution in every training step. This problem is known as internal covariate shift.

  The Batch Normalization layer forces the input of every layer to have approximately the same distribution in every training step by following below process during training time:
  - Calculate the mean and variance of the layers input.
  - Normalize the layer inputs using the previously calculated batch statistics.
  - Scale Layer scales and shifts in order to obtain the output of the layer.

  This makes the learning of layers in the network more independent of each other and allows you to be care free about weight initialization, works as regularization in place of dropout, and other regularization techniques.

The architecture above provides nonlinearities and preservation of dimension that help to improve the robustness of the network and control over fitting.

4.2.2. Human Presence Detection Network Output

From the input image model first extracts feature maps, overlays them with a WxH grid and at each cell computes K pre-computed bounding boxes called anchors. Each bounding box has the following:

- Four scalars (x, y, w, h)
  - A confidence score (Pr(Object)xIOU)
  - C° conditional class probability
- The current model architecture has a fixed output of *WxHxK(4+1+C)* where:
  - W , H = Grid Size
  - K = Number of Anchor boxes
  - C = Number of classes for which you want detection
- The model has a total of 672 output values which are derived from the following:
  - 4 x 4 grid
    - 7 anchor boxes per grid
      - 6 values per anchor box. It consists of:
        - 4 bounding box coordinates (x, y, w, h)
        - 1 class probability
        - 1 confidence score

So in total, 4 x 4 x 7 x 6 = 672 output values.

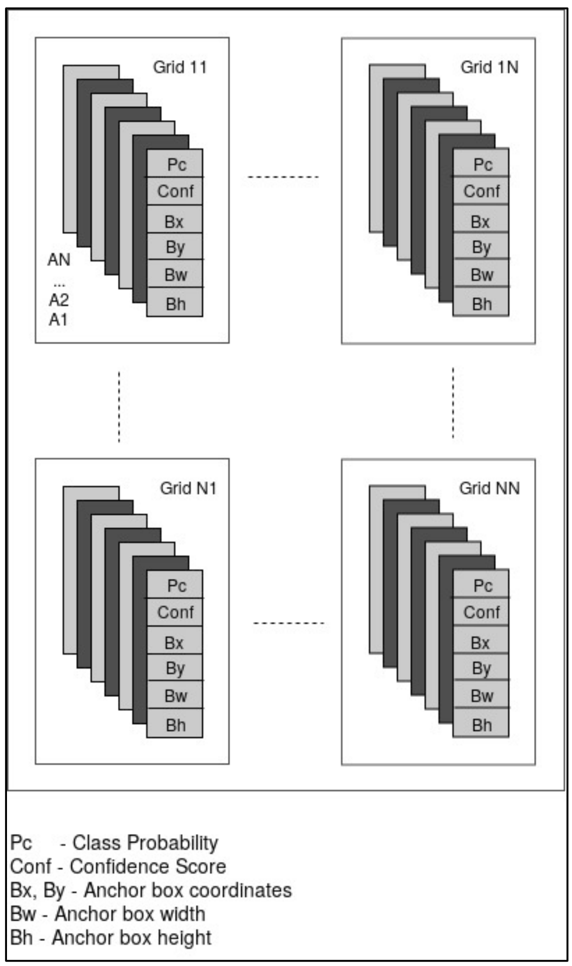

Figure 4.3. Model Output Format

4.2.2.1. Model Output Format on Hardware

- In hardware, the Human Presence Detection demo works based on confidence score. Other output values like class probability and bbox coordinates are byproduct which are not used at all.
- If the last layer in the network is Convolution, CNN IP supports partial output processing based on given filter range. SensAI tool provides option to specify the required filter range from the convolution layer output. This also results in hardware performance improvement.
- In Human Presence demo, the last convolution layer has 42 filters as described in Neural Network Architecture section**.** Out of 42, the first seven filters give class probability values; the next seven are for confidence score, and the rest for bbox coordinates.
- By configuring *output depth range* as shown in Figure 6.4, CNN only gives 112 confidence values as output.

4.2.3. Training Code Overview

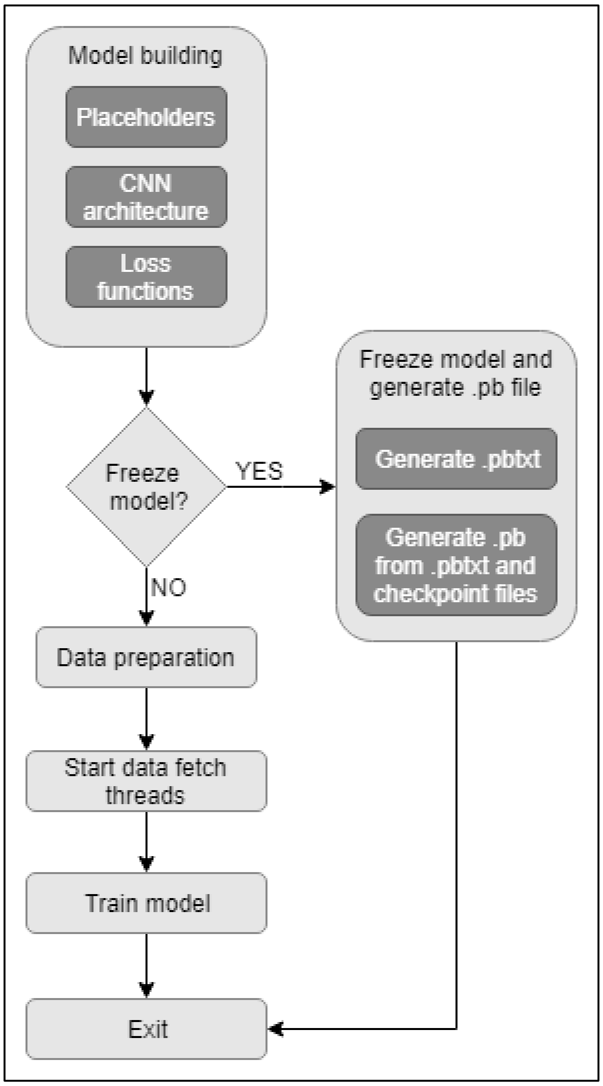

Figure 4.4. Training Code Flow Diagram

Training code is divided into the following parts:

- Model Config
- Model Building
- Model Freezing
- Data Preparation
- Training for Overall Execution Flow

Details of each can be found in subsequent sections.

4.2.3.1. Model Config

The design uses the Kitti dataset and SqueezeDet model. *kitti_squeezeDet_config.py* maintains all the configurable parameters for the model. Below is a summary of the configurable parameters:

- Image size
  - Change mc.IMAGE_WIDTH and mc.IMAGE_HEIGHT to configure Image size (width and height) in *src/config/kitti_squeezeDet_config.py*

```
mc.IMAGE_WIDTH          = 64 #224
mc.IMAGE_HEIGHT         = 64 #224
```

Figure 4.5. Code Snippet – Input Image Size Config

  - Since there are four pooling layers, grid dimension would be H = mc.IMAGE_WIDTH/(2 ^ 4) and W = mc.IMAGE_ HEIGHT/(2 ^ 4). Update grid size anchors per grid in set_anchors() in *src/config/kitti_squeezeDet_config.py*, that is if image size is 64 x 64, H = 64 / 16 = 4 and W = H = 64 / 16 = 4.

```
H, W, B = 4, 4, 7 # 64/16=4, 7 anchors
div_scale = 2.0 * 3.5 # 224/64=3.5
```

Figure 4.6. Code Snippet – Input Image Size Config (Grid Sizes)

- Batch size
  - Change mc.BATCH_SIZE in *src/config/kitti_squeezeDet_config.py* to configure batch size.

```
mc.BATCH_SIZE           = 20
```

Figure 4.7. Code Snippet – Batch Image Size Config

- Anchors per Grid
  - Change mc.ANCHOR_PER_GRID in *src/config/kitti_squeezeDet_config.py* to configure anchors per grid.

```
mc.ANCHOR_BOX           = set_anchors(mc)
mc.ANCHORS              = len(mc.ANCHOR_BOX)
mc.ANCHOR_PER_GRID      = 7
```

Figure 4.8. Code Snippet – Anchors per Grid Config #1

  - Change hard coded anchors per grid in set_anchors() in *src/config/kitti_squeezeDet_config.py*. Here, B (value 7) indicates anchors per grid.

```
def set_anchors(mc):
# H, W, B = 14, 14, 7
  H, W, B = 4, 4, 7 # 64/16=4, 7 anchors
  div_scale = 2.0 * 3.5 # 224/64=3.5
```

Figure 4.9. Code Snippet – Anchors per Grid Config #2

- anchor_shapes variable of set_anchors() in *src/config/kitti_squeezeDet_config.py* indicates anchors width and heights. Update it based on anchors per grid size changes.

```
anchor_shapes = np.reshape(
    [np.array(
        [
     [int(368./div_scale), int(368./div_scale)],
     [int(276./div_scale), int(276./div_scale)],
     [int(184./div_scale), int(184./div_scale)],
     [int(138./div_scale), int(138./div_scale)],
         [int( 92./div_scale), int( 92./div_scale)],
         [int( 69./div_scale), int( 69./div_scale)],
     [int( 46./div_scale), int( 46./div_scale)]])] * H * W,
    (H, W, B, 2)
)
```

Figure 4.10. Code Snippet – Anchors per Grid Config #3

- Training parameters
  - Other training related parameters like learning rate, loss parameters, and different thresholds can be configured from *src/config/kitti_squeezeDet_config.py*.

```
mc.WEIGHT_DECAY          = 0.0001
mc.LEARNING_RATE         = 0.01
mc.DECAY_STEPS           = 10000
mc.MAX_GRAD_NORM         = 1.0
mc.MOMENTUM              = 0.9
mc.LR_DECAY_FACTOR       = 0.5

mc.LOSS_COEF_BBOX        = 5.0
mc.LOSS_COEF_CONF_POS    = 75.0
mc.LOSS_COEF_CONF_NEG    = 100.0
mc.LOSS_COEF_CLASS       = 1.0

mc.PLOT_PROB_THRESH      = 0.4
mc.NMS_THRESH            = 0.4
mc.PROB_THRESH           = 0.005
mc.TOP_N_DETECTION       = 10

mc.DATA_AUGMENTATION     = True
mc.DRIFT_X               = 150
mc.DRIFT_Y               = 100
mc.EXCLUDE_HARD_EXAMPLES = False
```

Figure 4.11. Code Snippet – Training Parameters

4.2.3.2. Model Building

SqueezeDet class constructor builds the model which is divided into the following sections:

- Forward Graph
- Interpretation Graph
- Loss Graph
- Train Graph
- Visualization Graph

Forward Graph

- Forward Graph consists of seven fire layers.
    - Each fire layers contains a 3 x 3 convolution layer with padding=*SAME* and stride=1, a batch normalization layer, ReLU layer and an optional max pool layer. Out of these three fire layers, fire 2, fire 4, and fire 6 layers do not use max pool.
- These seven fire layers are followed by a 3 x 3 convolution layer with padding=*SAME* and stride=1.
- Filter sizes of each convolutional blocks is mentioned in Table 4.1, which can be configured by changing the values of *depth* shown in Figure 4.12.

```
depth = [16, 16, 32, 32, 32, 44, 48]  # 32KB for activation,  96KB for program #2 (n3)

####################################################################
# Quantization layers
####################################################################
if True: # 16b weight (no quant); 8b activation
    fl_w_bin = 8
    fl_a_bin = 8
    ml_w_bin = 8
    ml_a_bin = 8
    sl_w_bin = 8
    # The last layer's activation (sl_a_bin) is always 16b

    min_rng =  0.0 # range of quanized activation
    max_rng =  2.0

    bias_on = False # no bias for T+
```

Figure 4.12. Code Snippet – Quantization Value Setting

```
fire1 = self._fire_layer('fire1', self.image_input, oc=depth[0], freeze=False, w_bin=fl_w_bin, a_bin=fl_a_bin,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire2 = self._fire_layer('fire2', fire1, oc=depth[1], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire3 = self._fire_layer('fire3', fire2, oc=depth[2], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire4 = self._fire_layer('fire4', fire3, oc=depth[3], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire5 = self._fire_layer('fire5', fire4, oc=depth[4], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire6 = self._fire_layer('fire6', fire5, oc=depth[5], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin, pool_en=False,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire7 = self._fire_layer('fire7', fire6, oc=depth[6], freeze=False, w_bin=ml_w_bin, a_bin=ml_a_bin,
                                                   min_rng=min_rng, max_rng=max_rng, bias_on=bias_on)
fire_o = fire7
```

Figure 4.13. Code Snippet – Forward Graph Fire Layers

```
self.preds = self._conv_layer('conv12', fire_o, filters=num_output, size=3, stride=1,
    padding='SAME', xavier=False, relu=False, stddev=0.0001, w_bin=sl_w_bin, bias_on=bias_on)
```

Figure 4.14. Code Snippet – Forward Graph Last Convolution Layer

- 8-bit quantization is performed on weights and activations in this model. Based on the value of *w_bin* and *a_bin*, it is decided whether or not you should perform quantization.

```
if w_bin == 1: # binarized conv
    self.model_params += [kernel]
    kernel_bin = binarize(kernel)
    tf.summary.histogram('kernel_bin', kernel_bin)
    conv = tf.nn.conv2d(inputs, kernel_bin, [1, stride, stride, 1], padding=padding, name='convolution')
    conv_bias = conv
elif w_bin == 8: # 8b quantization
    kernel_quant = lin_8b_quant(kernel)
    tf.summary.histogram('kernel_quant', kernel_quant)
    conv = tf.nn.conv2d(inputs, kernel_quant, [1, stride, stride, 1], padding=padding, name='convolution')
    self.model_params += [kernel]

    if bias_on:
        biases = _variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        biases_quant = lin_8b_quant(biases)
        tf.summary.histogram('biases_quant', biases_quant)
        self.model_params += [biases]
        conv_bias = tf.nn.bias_add(conv, biases_quant, name='bias_add')
    else:
        conv_bias = conv
else:
    conv = tf.nn.conv2d(inputs, kernel, [1, stride, stride, 1], padding=padding, name='convolution')
    self.model_params += [kernel]
    if bias_on:
        biases = _variable_on_device('biases', [filters], bias_init, trainable=(not freeze))
        self.model_params += [biases]
        conv_bias = tf.nn.bias_add(conv, biases, name='bias_add')
    else:
        conv_bias = conv
```

Figure 4.15. Code Snippet – Quantization Layer

Interpretation Graph

- The Interpretation Graph consists of the following sub-blocks:
    - interpret_output
      As mentioned in Figure 4.3, model output is 4 x 4 x 42. There are 42 channels in the last layers which contain probability for the class, confidence score and bounding boxes values.
      This block interprets output from network and extracts predicted class probability, confidence score and bounding box values. From training code output value processing perspective, for each grid:
        - First N values (0:N-1) contains probabilities. Where N is number of anchor boxes. For N = 7, this ranges from 0 to 6 (including 6).
        - Next N values (N:2N – 1) contains confidence score. Where N is number of anchor boxes. For N = 7, this ranges from 7 to 13 (including 13).
        - Last 2N * 4 values contain bounding boxes information. Where N is number of anchor boxes. For N = 7, this ranges from 14 to 41 (including 41).

      The code below shows how the output from conv12 layer (4d array of batch size x 4 x 4 x 42) is sliced with proper indexes to get all values of probability, confidence, and coordinates.

```
# probability
num_class_probs = mc.ANCHOR_PER_GRID*mc.CLASSES
print ('ANCHOR_PER_GRID:', mc.ANCHOR_PER_GRID)
print ('CLASSES:', mc.CLASSES)
print ('preds2:', preds[:, :, :, :num_class_probs])
print ('ANCHORS:', mc.ANCHORS)

self.pred_class_probs = tf.reshape(
    tf.nn.softmax(
        tf.reshape(
            preds[:, :, :, :num_class_probs],
            [-1, mc.CLASSES]
        )
    ),
    [mc.BATCH_SIZE, mc.ANCHORS, mc.CLASSES],
    name='pred_class_probs'
)

# confidence
num_confidence_scores = mc.ANCHOR_PER_GRID+num_class_probs
self.pred_conf = tf.sigmoid(
    tf.reshape(
        preds[:, :, :, num_class_probs:num_confidence_scores],
        [mc.BATCH_SIZE, mc.ANCHORS]
    ),
    name='pred_confidence_score'
)

# bbox_delta
self.pred_box_delta = tf.reshape(
    preds[:, :, :, num_confidence_scores:],
    [mc.BATCH_SIZE, mc.ANCHORS, 4],
    name='bbox_delta'
)
```

Figure 4.16. Code Snippet – Interpret Output Graph

For confidence score, value should be between 0 and 1, so sigmoid is used.

For predicting the class probabilities, there is a vector of NUM_CLASS values at each bounding box. Applying softmax makes it a better probability distribution.

- Bbox - This block calculates bounding boxes based on anchor box and predicated bounding boxes.
- IOU – This block calculates Intersection Over Union for detected bounding boxes and actual bounding boxes.
- Probability – This block calculates detection probability and object class.

Loss Graph

- This block calculates the different types of losses which need to be minimized. There are three types of losses which are considered for calculation:
  - Class Probability

    The class loss function is just cross-entropy loss for classification for each box to do classification (predicted class versus actual class), as you would for image classification.

```
with tf.variable_scope('class_regression') as scope:
  # cross-entropy: q * -log(p) + (1-q) * -log(1-p)
  # add a small value into log to prevent blowing up
  self.class_loss = tf.truediv(
      tf.reduce_sum(
          (self.labels*(-tf.log(self.pred_class_probs+mc.EPSILON))
          + (1-self.labels)*(-tf.log(1-self.pred_class_probs+mc.EPSILON)))
          * self.input_mask * mc.LOSS_COEF_CLASS),
      self.num_objects,
      name='class_loss'
  )
  tf.add_to_collection('losses', self.class_loss)
```

Figure 4.17. Code Snippet – Class Loss

  - Bounding Box
    This loss is regression of the scalars for the anchors.

```
with tf.variable_scope('bounding_box_regression') as scope:
  self.bbox_loss = tf.truediv(
      tf.reduce_sum(
          mc.LOSS_COEF_BBOX * tf.square(
              self.input_mask*(self.pred_box_delta-self.box_delta_input))),
      self.num_objects,
      name='bbox_loss'
  )
  tf.add_to_collection('losses', self.bbox_loss)
```

Figure 4.18. Code Snippet – Bbox Loss

  - Confidence Score
    To obtain meaningful confidence score, each box's predicted value is regressed against the Intersection over Union of the real and the predicted box. During training, compare the ground truth bounding boxes with all anchors and assign them to the anchors that have the largest overlap (IOU) with each of them.

    The reason being is to select the *closest* anchor to match the ground truth box such that the transformation needed is reduced to minimum. Equation evaluates to 1 if the k-th anchor at position-(i, j) has the largest overlap with a ground truth box, and to 0 if no ground truth is assigned to it. This way, include only the loss generated by the *responsible* anchors.

    As there can be multiple objects per image, normalize the loss by dividing it by the number of objects (*self.num_objects*).

```
with tf.variable_scope('confidence_score_regression') as scope:
  input_mask = tf.reshape(self.input_mask, [mc.BATCH_SIZE, mc.ANCHORS])
  self.conf_loss = tf.reduce_mean(
      tf.reduce_sum(
          tf.square((self.ious - self.pred_conf))
          * (input_mask*mc.LOSS_COEF_CONF_POS/self.num_objects
             +(1-input_mask)*mc.LOSS_COEF_CONF_NEG/(mc.ANCHORS-self.num_objects)),
          reduction_indices=[1]
      ),
      name='confidence_loss'
  )
  tf.add_to_collection('losses', self.conf_loss)
  tf.summary.scalar('mean iou', tf.reduce_sum(self.ious)/self.num_objects)
```

Figure 4.19. Code Snippet – Confidence Loss

Train Graph

- This block is responsible for training the model with momentum optimizer to reduce all losses.

Visualization Graph

- This provides visualization of detected results.

4.3. Training from Scratch and/or Transfer Learning

To train the machine:

1. Go to the top/root directory of the Lattice training code from command prompt.

   The Model works on 64 x 64 input resolution for training.

   Current human detection training code uses mean = 0 and scale = 1/128 (0.0078125) in pre-processing step. Mean and scale can be changed in training code *@src/dataset/imdb.py* as shown in Figure 4.20.

```
v = np.where(v <= 255 - add_v, v + add_v, 255)
final_hsv = cv2.merge((h, s, v))
im = cv2.cvtColor(final_hsv, cv2.COLOR_HSV2BGR)

im -= mc.BGR_MEANS # <---------------------------
im /= 128.0 # to make input in the range of [0, 2)
orig_h, orig_w, _ = [float(v) for v in im.shape]
```

Figure 4.20. Training Code Snippet for Mean and Scale

The dataset path can be set in the training code *@src/dataset/kitti.py* and can be used in combination with the --*data_path* option while triggering training using *train.py* to get the desired path. For example, you can have *<data_path>/training/images* and *<data_path>/training/labels*.

```
def __init__(self, image_set, data_path, mc):
  imdb.__init__(self, 'kitti_'+image_set, mc)
  self._image_set = image_set
  self._data_root_path = data_path
  self._image_path = os.path.join(self._data_root_path, 'training', 'images')
  self._label_path = os.path.join(self._data_root_path, 'training', 'labels')
  self._classes = self.mc.CLASS_NAMES
```

Figure 4.21. Training Code Snippet for Dataset Path

2. Create a train.txt.

```
$ cd data/humandet/
$ python dataset_create.py
```

```
k$ python dataset_create.py
k$ _
```

Figure 4.22. Create File for Dataset train.txt

Notes:

- train.txt – file name of dataset images.
- image_set – train (ImageSets/train.txt)
- data_path – $ROOT/data/humandet/.
    - Images – $ROOT/data/humandet/images
    - Annotations – $ROOT/data/humandet/labels

3. Modify the training script.

   Training script at *@scripts/train.sh* is used to trigger training. Figure 4.23 shows the input parameters which can be configured.

```
python ./src/train.py \
  --dataset=KITTI \
  --pretrained_model_path=$PRETRAINED_MODEL_PATH \
  --data_path=$TRAIN_DATA_DIR \
  --image_set=train \
  --train_dir="$TRAIN_DIR/train" \
  --net=$NET \
  --summary_step=100 \
  --checkpoint_step=500 \
  --max_steps=2000000 \
  --gpu=$GPUID
```

Figure 4.23. Training Input Parameter

- $TRAIN_DATA_DIR – dataset directory path. */data/humandet* is an example.
- $TRAIN_DIR – log directory where checkpoint files are generated while model is training.
- $GPUID – gpu id. If the system has more than one gpu, it indicates the one to use.
- --summary_step – indicates at which interval loss summary should be dumped.
- --checkpoint_step – indicates at which interval checkpoints is created.
- --max_steps – indicates the maximum number of steps for which the model is trained.

4. Execute the *run* command script which starts training.

```
k$ ./run
Using TensorFlow backend.
self.preds: Tensor("conv12/convolution:0", shape=(20, 4, 4, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 4, 4, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 112
```

Figure 4.24. Execute Run Script

5. Start TensorBoard.

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

For example: tensorboard –logdir='./logs/'

6. Open the local host port on your web browser.

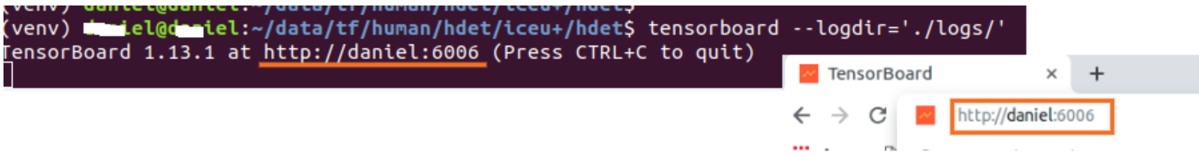

Figure 4.25. TensorBoard – Generated Link

7. Check the training status on TensorBoard.

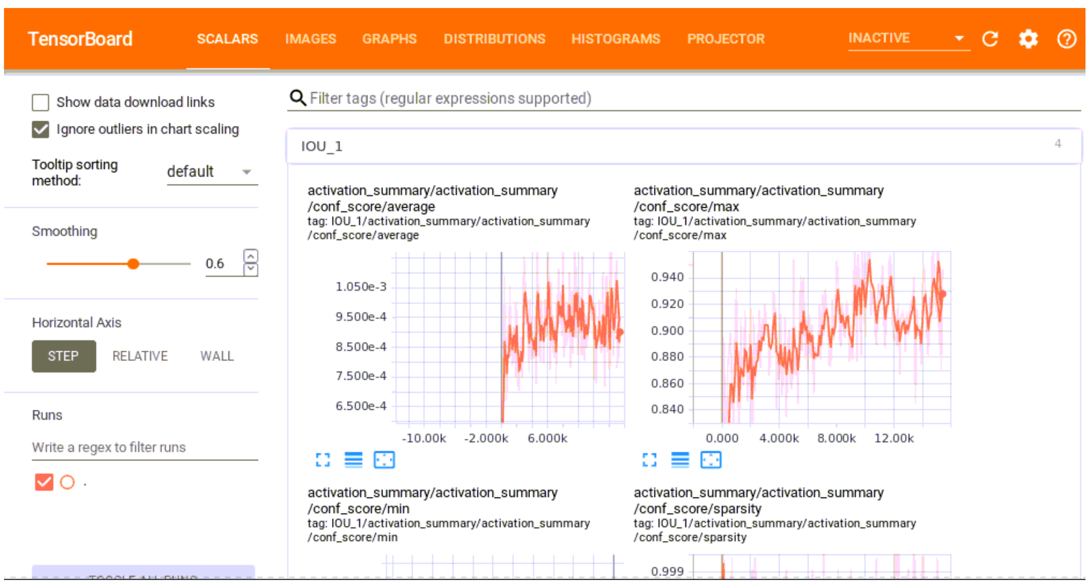

Figure 4.26. TensorBoard

Figure 4.27 shows the image menu of TensorBoard.

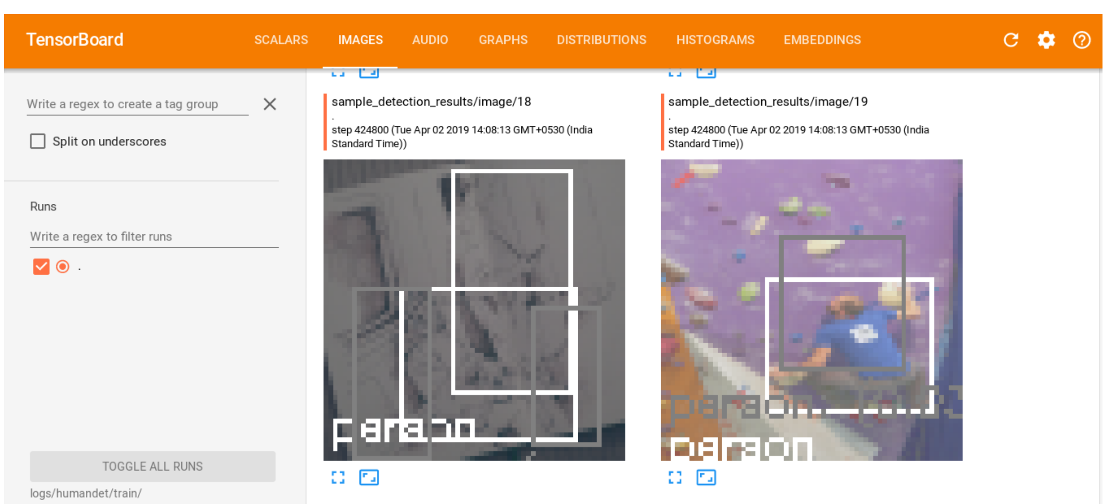

Figure 4.27. Image Menu of TensorBoard

8. Check if the *checkpoint*, *data*, *meta*, *index*, and *events* (if using TensorBoard) files are created at the log directory. These files are used for creating the frozen file (*.pb).

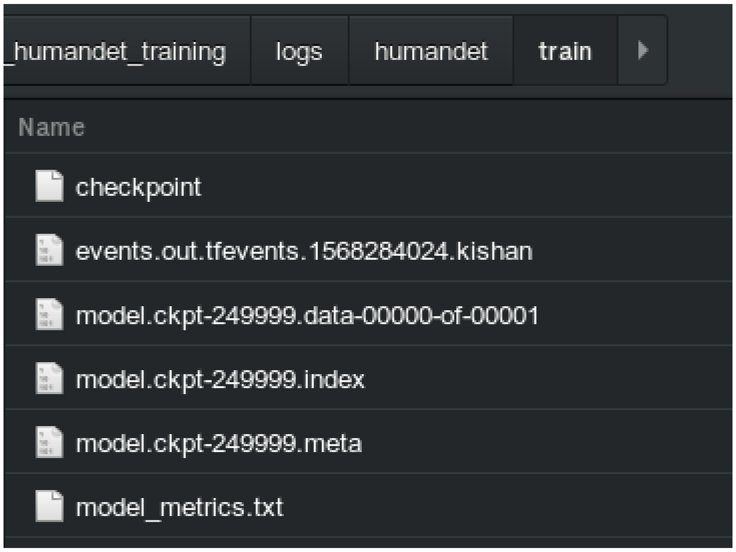

Figure 4.28. Example of Checkpoint Data Files at Log Folder

5. Creating Frozen File

This section describes the procedure for freezing the model, which is aligned with the Lattice SensAI tool. Perform the steps below to generate the frozen protobuf file:

5.1. Generating the frozen .pb File

Generate .pb file from latest checkpoint using below command from the training code's root directory.

```
$ python src/genpb.py –ckpt_dir=”<log directory>” --freeze
```

For example, *python src/genpb.py –ckpt_dir './logs/humandet/train/' –freeze.*

```
earth:~/human_presence-2.1$ python src/genpb.py --ckpt_dir=logs/humandet/train/ --freeze
genrating pbtxt
self.preds: Tensor("conv12/convolution:0", shape=(20, 4, 4, 42), dtype=float32, device=/device:GPU:0)
ANCHOR_PER_GRID: 7
CLASSES: 1
preds2: Tensor("interpret_output/strided_slice:0", shape=(20, 4, 4, 7), dtype=float32, device=/device:GPU:0)
ANCHORS: 112
Using checkpoint: ./model.ckpt-249999
saved pbtxt at checkpoint direcory Path
('inputShape shape', [1, 64L, 64L, 3L])
```

Figure 5.1. pb File Generation from Checkpoint

Figure 5.2 shows the generated .pb file.

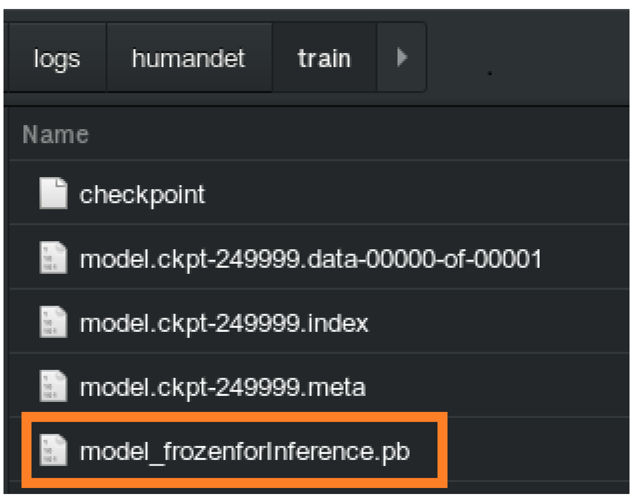

Figure 5.2. Frozen pb File

6. Creating Binary File with SensAI

This chapter describes how to generate binary file using the Lattice SensAI version 2.1 program.

Figure 6.1. SensAI Home Screen

To create the project in SensAI tool:

1. Click **File > New**.
2. Enter the following settings:
    - **Project Name**
    - **Framework – TensorFlow**
    - **Class – CNN**
    - **Device – UltraPlus**
3. Click **Network File** and select the network (PB) file.

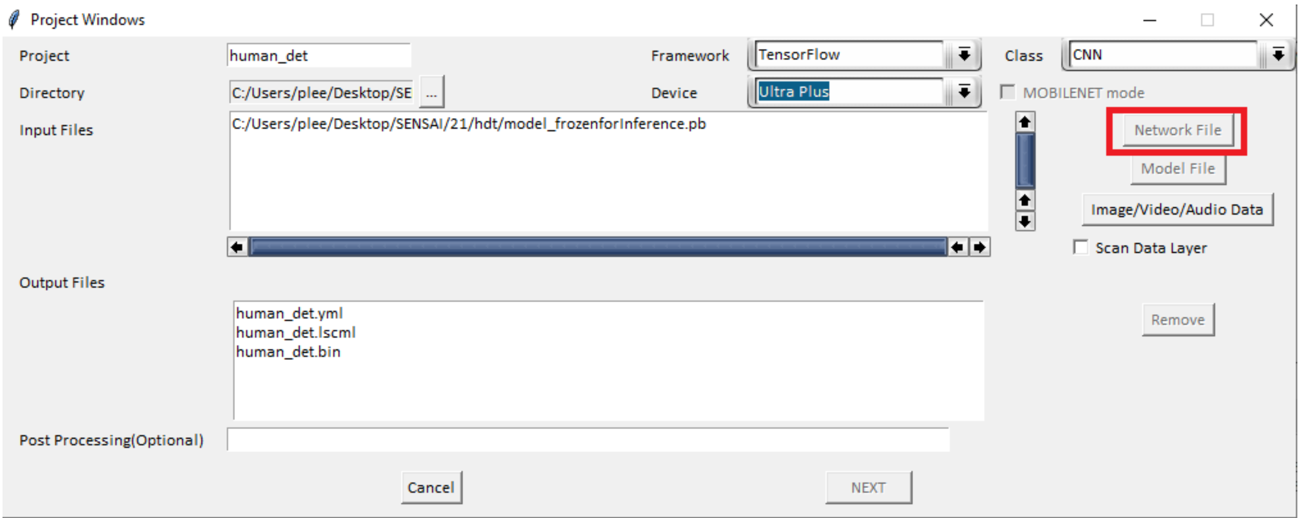

Figure 6.2. SensAI –Network File Selection

4. Click **Image/Video/Audio Data** and select the image input file.

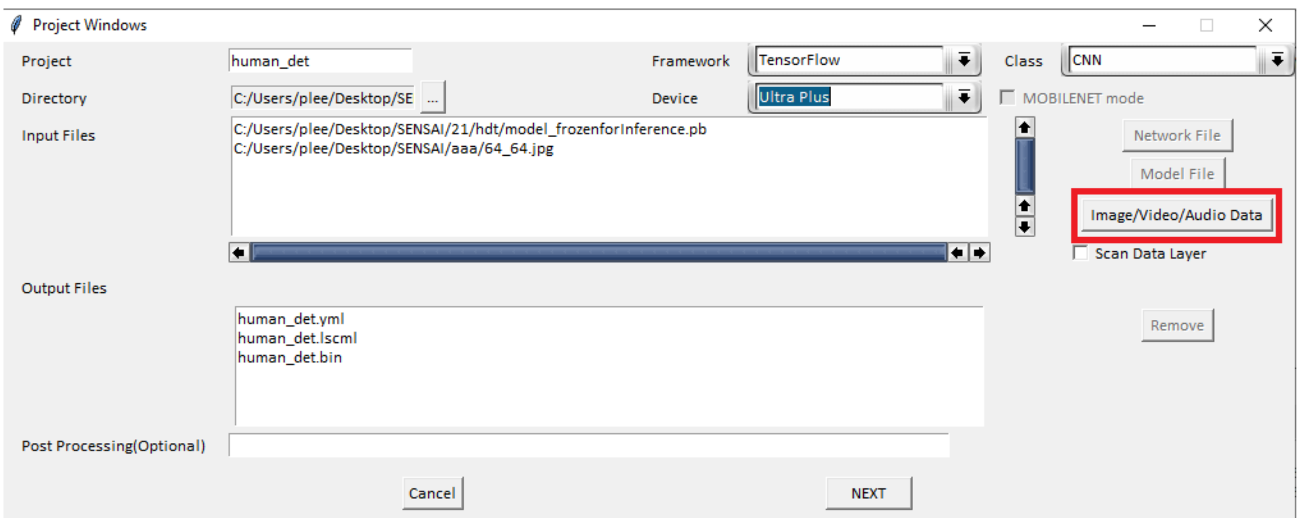

Figure 6.3. SensAI –Image Data File Selection

5. Click **NEXT**.
6. Configure your project settings.

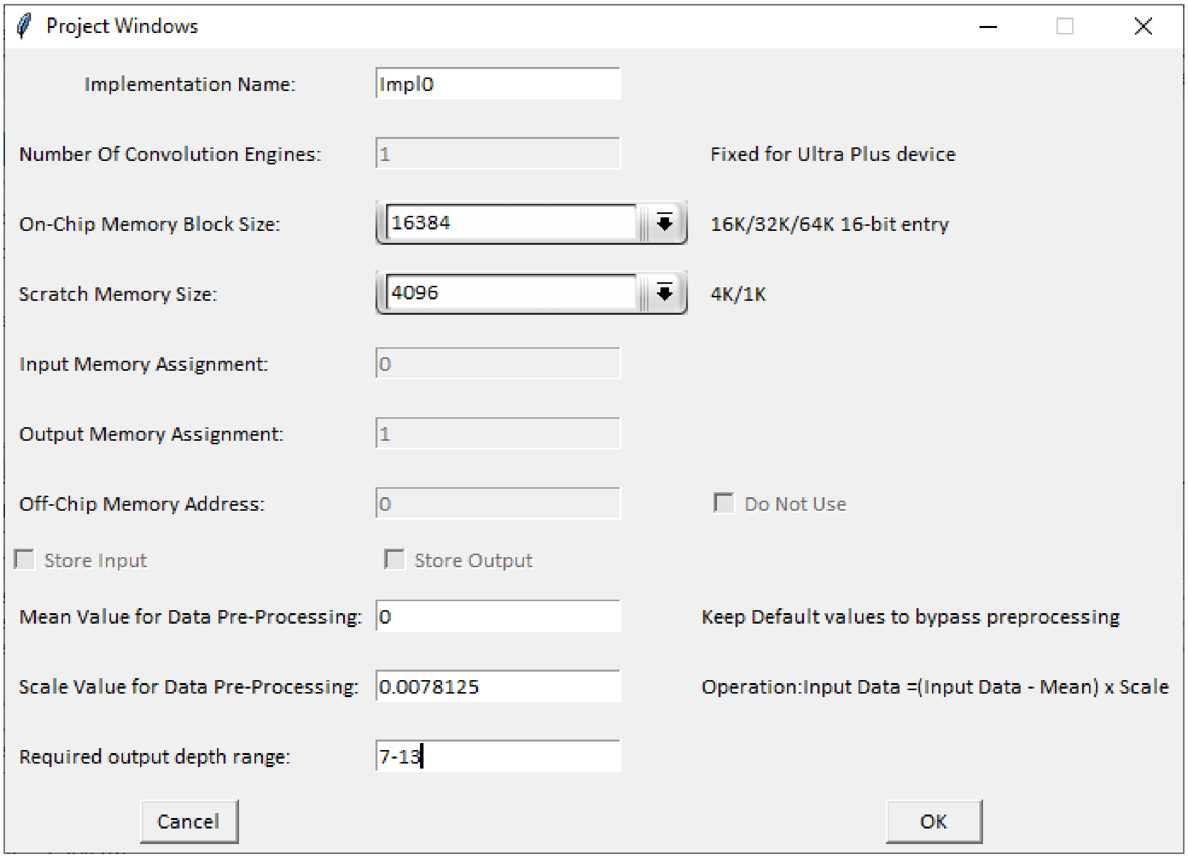

Figure 6.4. SensAI – Project Settings

7. Click **OK** to create project.
8. Double-click **Analyze**.

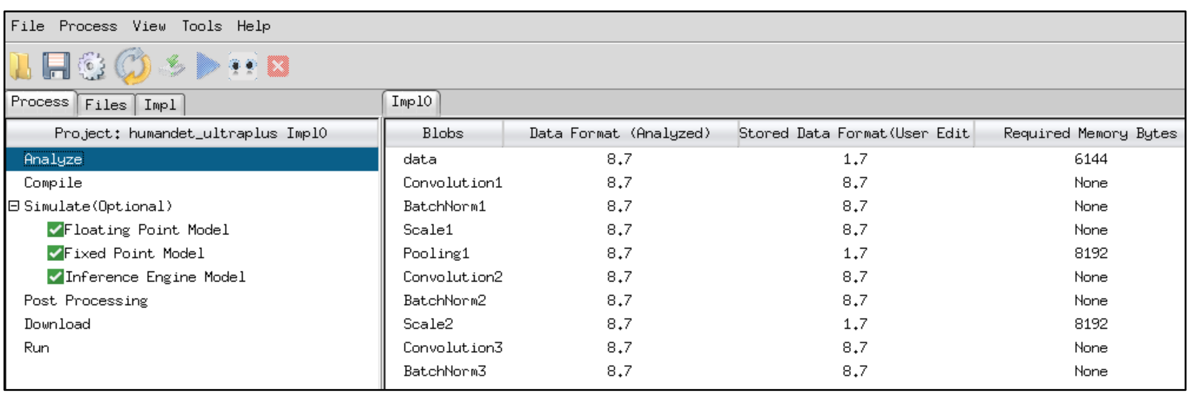

Figure 6.5. SensAI – Analyze Project

9. Double-click **Compile** to generate the Firmware file.

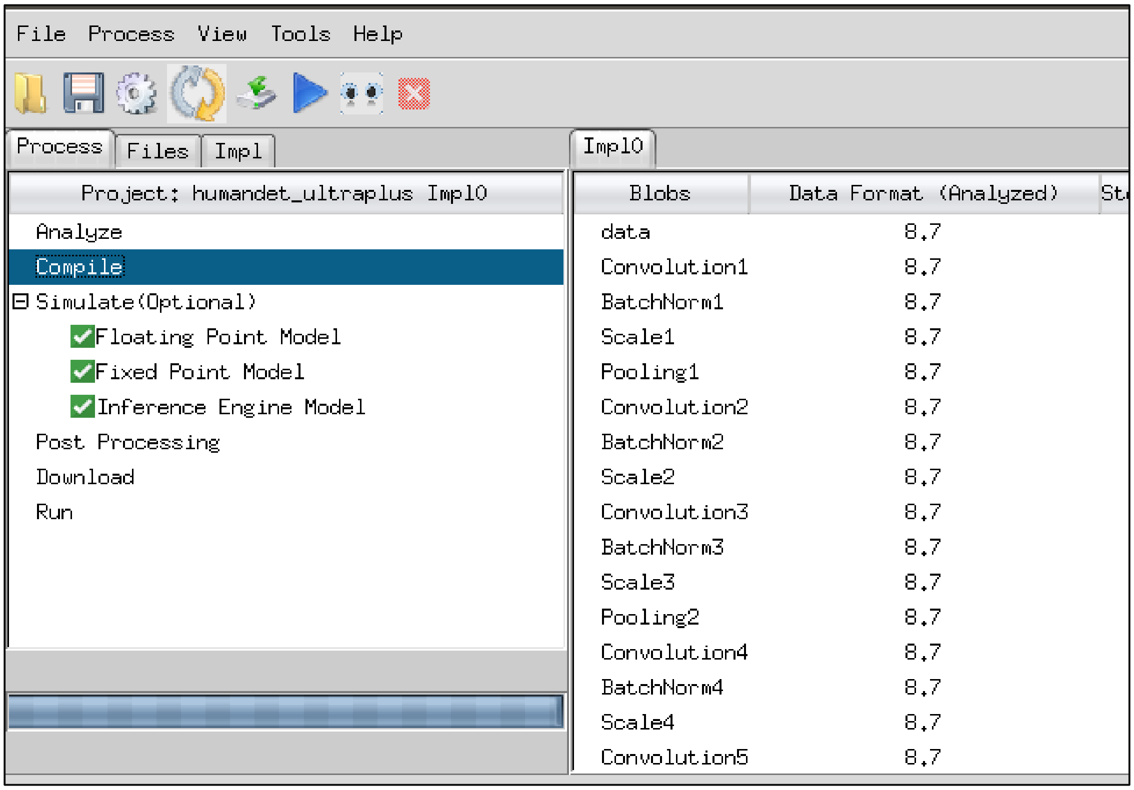

Figure 6.6. Compile Project

7. Hardware (RTL) Implementation

7.1. Top Level Information

7.1.1. Block Diagram

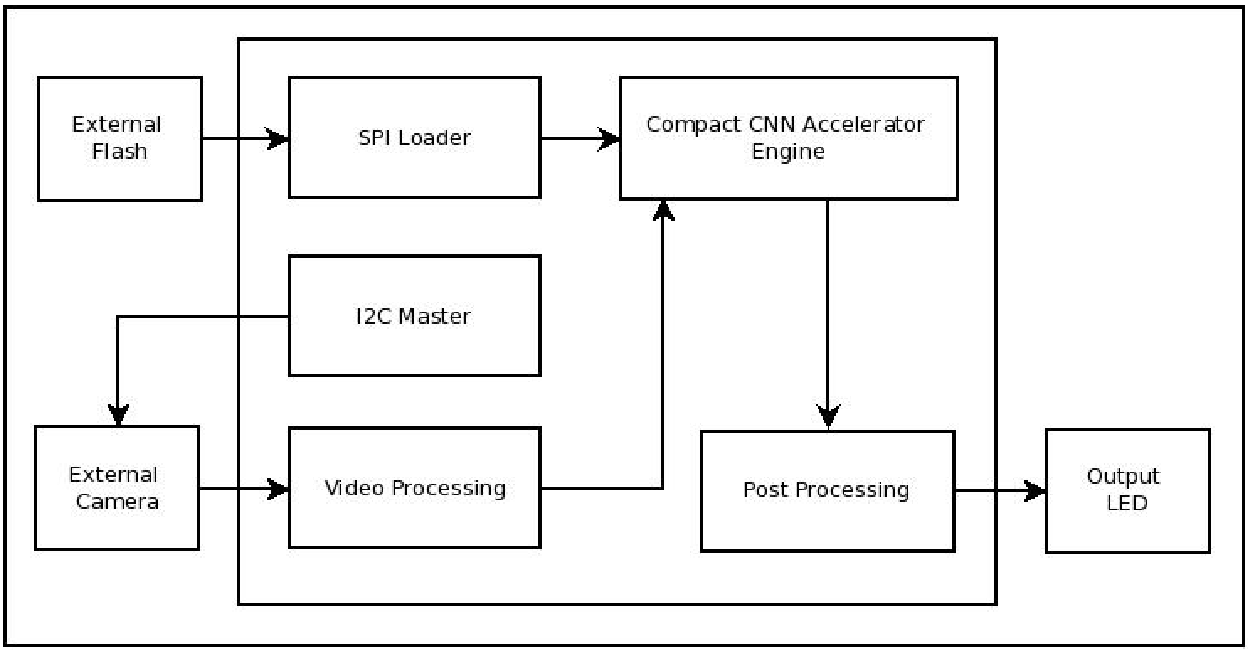

Figure 7.1. Top Level Block Diagram Human Presence Detection iCE40

7.1.2. Overall Operational Flow

- The external camera is configured via I^{2}C Master block for desired camera settings right after the system is powered up. The camera captures the real-time image data and sends it to the iCE40 Ultra Plus device.
- The input image data is passed through Video Processing module which performs Crop and Downscale operation which makes input resolution compatible for CNN model and maintains the aspect ratio. Current CNN model requires 64 x 64 input resolution.
- CNN IP uses the image data with the firmware file from the external SPI Flash and does the inference and generates output.
- CNN output is passed to Post Processing module which processes output to find out the maximum value. This value is utilized to take decision of driving the LED output.
- In this demo, there are six LED lights with potential to turn. According to the parameter MIRROR_MODE configuration, the demo has two following output representations:
  - Configuration 1 (MIRROR_MODE = 0)
    - Human Presence is detected and given as output in form of six LEDs. From top to bottom:
      - LED D1 represents human detection in Upper Left of the screen,
      - LED D2 represents human detection in Upper Right of the screen,
      - LED D3 represents human detection in Lower Left of the screen,
      - LED D4 represents human detection in Lower Right of the screen,
      - LED D5 represents human detection in Center of the camera,
      - LED D6 represents human detection in the Full image.

- Configuration 2 (MIRROR_MODE = 1)
  - Human Presence is detected and given as output in form of one LED:
    - LED D5 represents human detection in the Full image.

7.1.3. Core Customization

Table 7.1. Core Parameters

Parameter	Default (Decimal)	Description
MIRROR_MODE	1	1 – Single LED output if Human Presence Detected 0 – Six LED output according to Human Presence Detected in any of the six zones.
BYTE_MODE	UNSIGNED	Configured for CNN input data layer width. It is to be kept according to the *Mean* parameter setting from software training. UNSIGNED – The data is directly passed to CNN input for unsigned 8-bit input data layer. SIGNED – 128 is subtracted from the data for signed 8-bit input data layer of CNN. DISABLED – Disable byte mode

7.2. Architectural Details

7.2.1. CNN Pre-Processing

7.2.1.1. MIRROR_MODE=0 (Zoning Enabled)

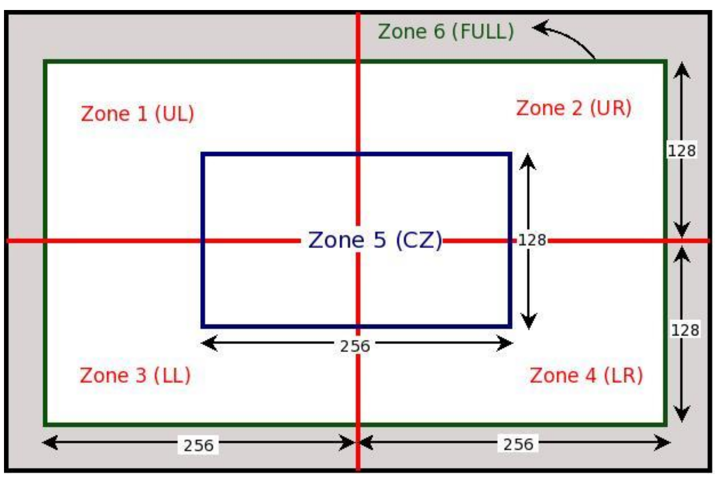

Figure 7.2. Image Zoning Enabled

- The camera output provides full frame image data which can be divided into six zones according to the value of frame zone counter. The Frame Zone counter is implemented in the top module as shown in Figure 7.3, and it counts from 0 to 5 (6 values) mapping each count value to each zone. By default, it is set to 0.
- When Frame Zone Counter is 0, the Upper Left of Image data is utilized for CNN input. Counter is incremented when CNN starts processing that Data. The Upper Right of image data is utilized for CNN input for counter value 1.

  The frame zone counter values are mapped to frame zones as follows:
  - count 0 – Upper Left Zone
  - count 1 – Upper Right Zone
  - count 2 – Lower Left Zone
  - count 3 – Lower Right Zone
  - count 4 – Center Zone
  - count 5 – Full Image Zone

```
always @(posedge clk or negedge resetn)
begin
    if(resetn == 1'b0)
    r_frame_sel <= 3'd0;
    else if((MIRROR_MODE == 1'b1) || (EN_SEQ == 1) || (FUSION_MODE == 1'b1))
    r_frame_sel <= 3'b101;
    else if(r_rd_done_d == 2'b10)
    r_frame_sel <= (r_frame_sel == 3'b101) ? 3'd0 : (r_frame_sel + 3'd1);
end
```

Figure 7.3. RTL logic – Zone Counter

7.2.1.2. Zoning and Downscaling (ice40_himax_video_process_128.v)

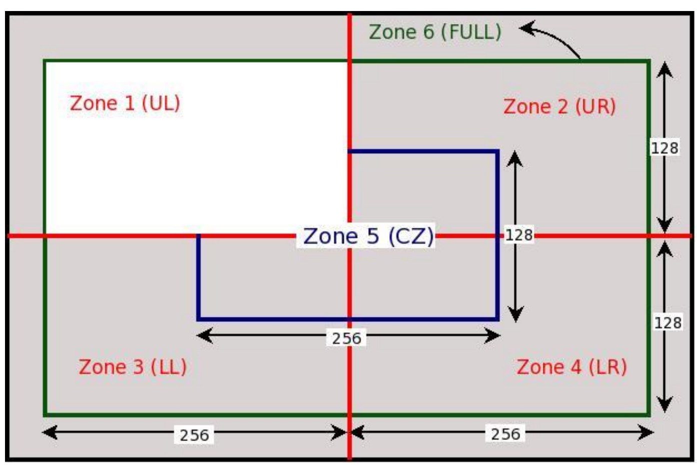

Figure 7.4. Masking for Zone 1

Image data values are streamed continuously from input camera serial interface.

The horizontal/vertical masking is performed on the input image to mask out the boundary area to make the image resolution in multiple of CNN input resolution (64 x 64). The pixel values of image under mask area are not considered valid. Masking makes the downscaling process easier.

7.2.1.3. Downscaling for Zones 1-5

- Mask values are set according to the current active image zone. For zones 1 to 5, masking produces image resolution of 256 x 128 from full image frame as shown in the Figure 7.4.
- As shown in Figure 7.4 when Zone 1 is active, whole image is masked apart from upper left 256 x 128 pixel block. 256 x 128 resolution image data is then downscaled to 64 x 64 resolution in video processing module as explained below.

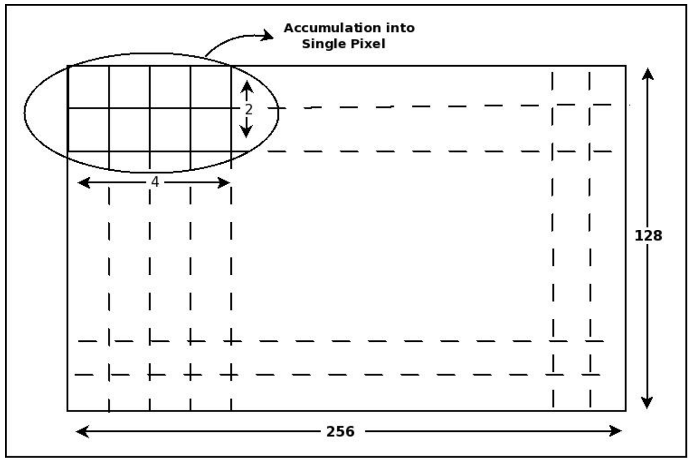

Figure 7.5. Downscaling Zones 1-5

- As shown in the Figure 7.5, Every 4 horizontal (256/64) and 2 vertical (128/64) pixel which make pixel grid of 4 x 2 are accumulated into a single pixel value to generate 64 x 64 resolution image for CNN input.
- The accumulated RGB pixel values are written into accumulation buffer. As every 64 values are written into the memory, Data from memory is read and transferred to CNN. This way, the 4096 (64 x 64) red, green, and blue pixel values are passed on to CNN input.

7.2.1.4. Downscaling for Zone 6 (Full Image Zone)

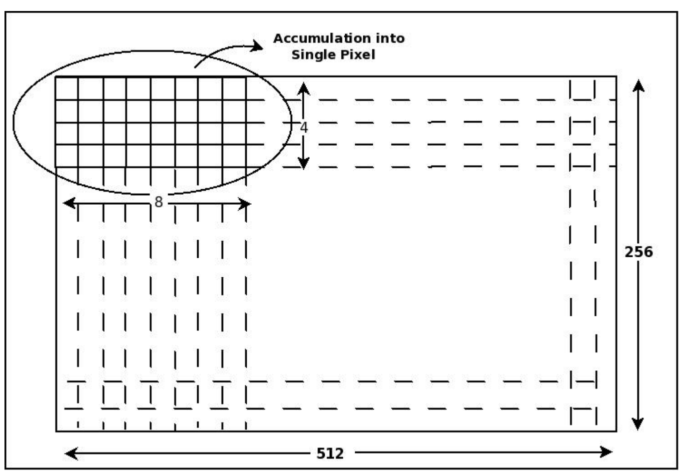

Figure 7.6. Downscaling Zone 6

- For zone 6, masking produces image resolution of 512 x 256 from full image frame as shown in Figure 7.6. The image data is then downscaled into 64 x 64 resolution.
- As shown in the Figure 7.6, Every 8 horizontal (512/64) and 4 vertical (256/64) pixel which make pixel grid of 8 x 4 are accumulated into a single pixel value to produce 64 x 64 resolution image for CNN input.
- The accumulated RGB pixel values are written into accumulation buffer. As every 64 values are written into the memory, data from memory is read and transferred to CNN. This way, the 4096 (64 x 64) red, green, and blue pixel values are passed to CNN.

7.2.1.5. MIRROR_MODE=1 (Zoning Disabled)

Figure 7.7. Image Zoning Disabled

- The Frame Zoning is disabled in this configuration. The camera output image data is not divided into different frame zones but it is treated as Full image Zone as shown in Figure 7.7.
- The Frame Zone counter which is common for both the MIRROR_MODE configurations is kept at full frame zone count value (that is count=5) as shown in the Figure 7.3.
- By default, the full frame zone (that is zone-6) is always enabled in the operation flow.
- The camera output image resolution is masked and downscaled according to zone-6 as described in Downscaling for Zone 6 (Full Image Zone) section.

7.2.2. CNN Post-Processing (humandet_post.v)

- CNN provides 112 output values as described in Model Output Format on Hardware section. These values are confidence score for 7 anchor boxes of each 4 x 4 grid (4 x 4 x 7 = 112). Post processing logic in this module is independent of number of values provided by CNN.
- The post processing logic finds out the maximum value from all the confidence values provided by CNN output using RTL logic as shown in Figure 7.8.

```
//==============================================================================
// Fixed point comparison - left value >= right value
//==============================================================================
function fixed_lte;
    input [15 : 0] lval;
input [15 : 0] rval;
begin
    fixed_lte = (!lval[15] && rval[15]                                  ) || // pos >= neg
            ( lval[15] ~^ rval[15] ? lval[14:0] >= rval[14:0] : 1'b0);   // same polarity, large is larger
                                                                  // diff polarity (neg, pos)
end endfunction
```

Figure 7.8. RTL Logic – Maximum CNN Value Calculation

- The calculated maximum value (**w_class0** signal) is passed to Top module. The maximum value is considered valid only if it is a positive value i.e. SIGN bit (Highest bit) is 0. The SIGN Bit is used to take decision of driving the LED as shown in below RTL logic.
- The frame zone counter count value determines which LED to drive.

```
always @(posedge clk)
begin
    if(r_comp_done_d[7] == 1'b1) begin
    if(MIRROR_MODE == 1'b1)
        r_det_vec<= {1'b0, !w_class0[15], 4'b0};
    else case(r_frame_sel)
        3'd0 :   r_det_vec[5] <= !w_class0[15];
        3'd1 :   r_det_vec[0] <= !w_class0[15];
        3'd2 :   r_det_vec[1] <= !w_class0[15];
        3'd3 :   r_det_vec[2] <= !w_class0[15];
        3'd4 :   r_det_vec[3] <= !w_class0[15];
        default: r_det_vec[4] <= !w_class0[15];
    endcase
    end
end
```

Figure 7.9. RTL Logic – Driving Output LED Logic[1]

```
end else if(EN_UPDUINO2)
begin: g_on_upduino2
    /* UL      */ assign oled[0] = r_det_vec[0] ? 1'bz : 1'b0;
    /* UR      */ assign oled[1] = r_det_vec[1] ? 1'bz : 1'b0;
    /* LL      */ assign oled[2] = r_det_vec[2] ? 1'bz : 1'b0;
    /* LR      */ assign oled[3] = r_det_vec[3] ? 1'bz : 1'b0;
    /* CZ      */ assign oled[4] = r_det_vec[4] ? 1'bz : 1'b0;
    /* FULL    */ assign oled[5] = r_det_vec[5] ? 1'bz : 1'b0;
```

Figure 7.10. RTL Logic – Driving Output LED Logic[2]

7.2.2.1. MIRROR_MODE=0 (Zoning Enabled)

From above RTL logic, LEDs are driven as following for frame zone counter values:

- Count 0 → LED 1 (UL) is ON (if Max CNN output value SIGN Bit = 0)
- Count 1 → LED 2 (UR) is ON (if Max CNN output value SIGN Bit = 0)
- Count 2 → LED 3 (LL) is ON (if Max CNN output value SIGN Bit = 0)
- Count 3 → LED 4 (LR) is ON (if Max CNN output value SIGN Bit = 0)
- Count 4 → LED 5 (Center) is ON (if Max CNN output value SIGN Bit = 0)
- Count 5 → LED 6 (Full) is ON (if Max CNN output value SIGN Bit = 0)

7.2.2.2. MIRROR_MODE=1 (Zoning Disabled)

The frame zone counter is assigned fixed value of count 4 which drives the LED 5 as follows:

- Count 4 → LED 5 (Center) is ON (if Max CNN output value SIGN Bit = 0)

8. Creating FPGA Bitstream File

This section provides the procedure for creating your FPGA bitstream file using Lattice Radiant Software.

Note: This reference design includes a Compact CNN IP that requires a license to be able to generate a bitstream. Lattice provides a 30-day evaluation license for this IP for those who want to evaluate the IP and reference design. You can obtain an evaluation license from the Lattice website Software Licensing page.

Lattice Radiant software version 1.1 is required to generate a bitstream along with a software license patch. You can obtain the software patch file from the Lattice website through Lattice Radiant 1.1 Software Patch.

To create the FPGA bitstream file:

1. Open Lattice Radiant Software.

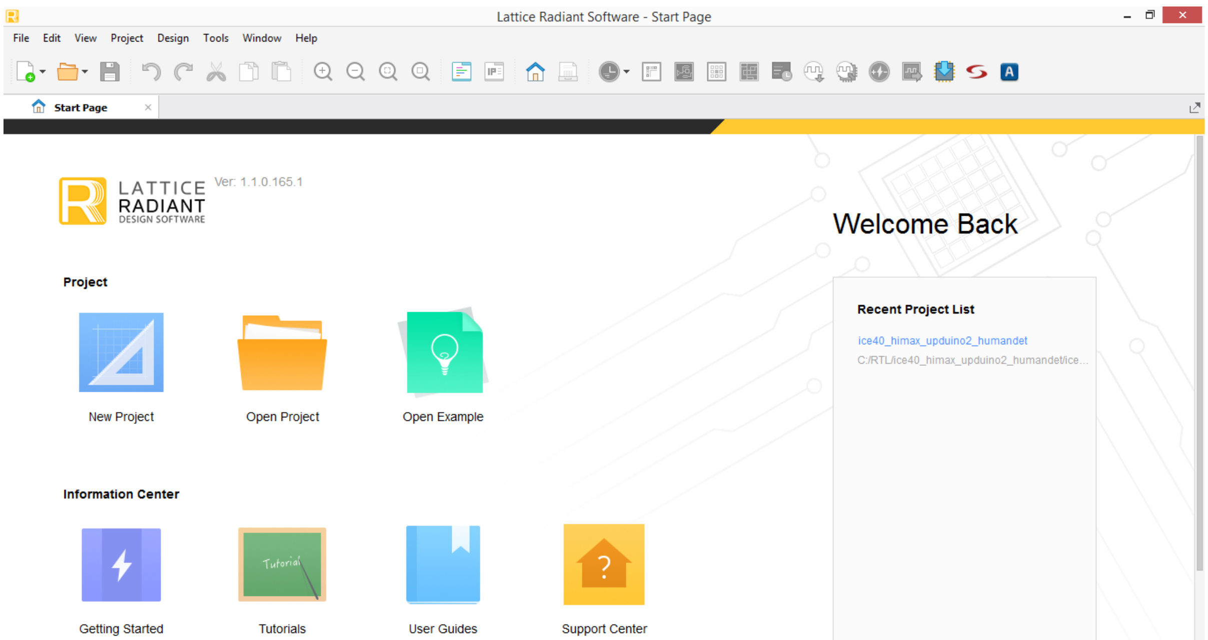

Figure 8.1. Radiant Software

2. Click **File > Open Project**.
3. Open the Radiant project file for iCE40 human presence detection RTL.

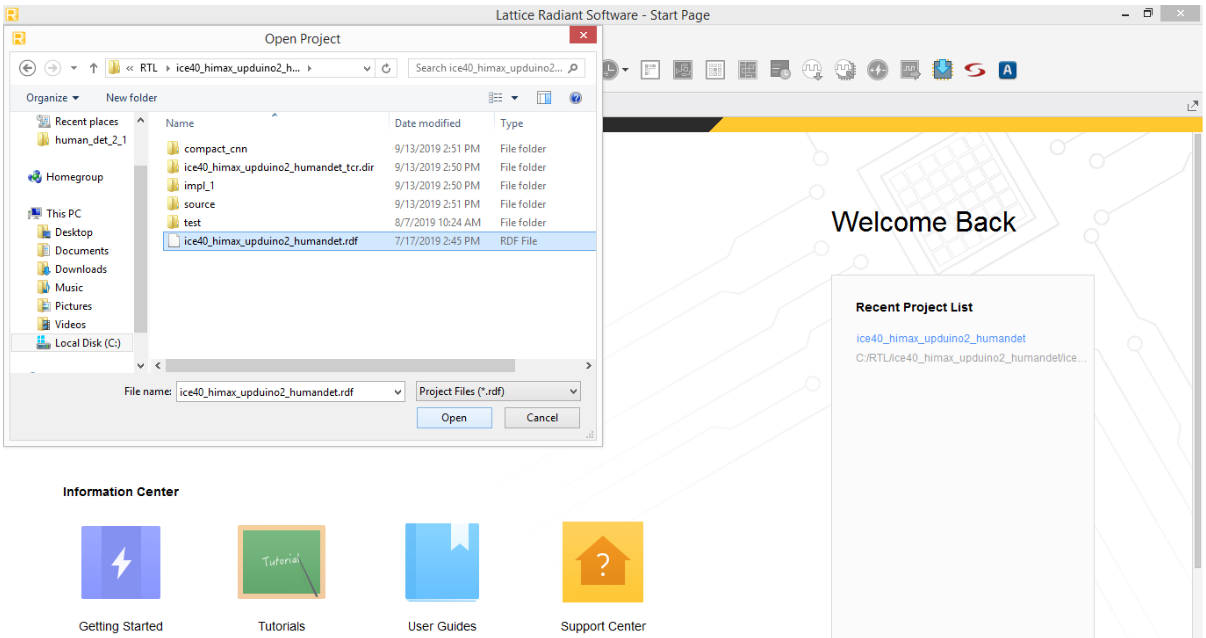

Figure 8.2. Radiant Software – Open Project

4. Click **Export** to generate the bit file.

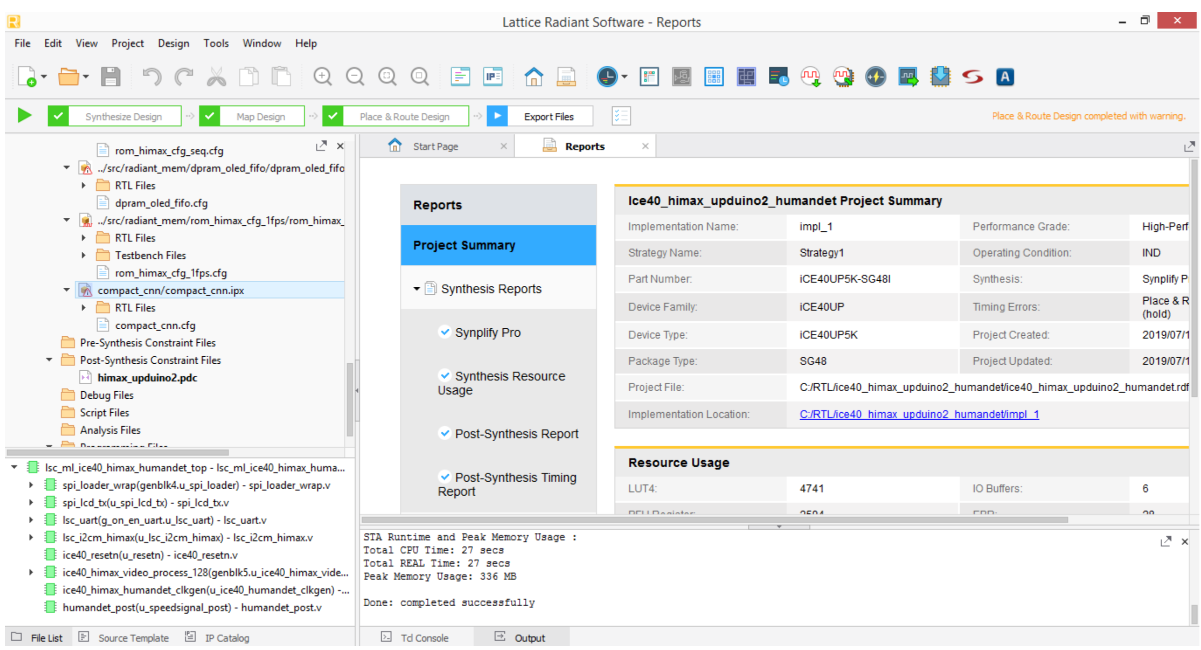

Figure 8.3. Radiant Software – Bitstream Generation

5. View the log message in **Export Reports** that indicates the generated bitstream path.

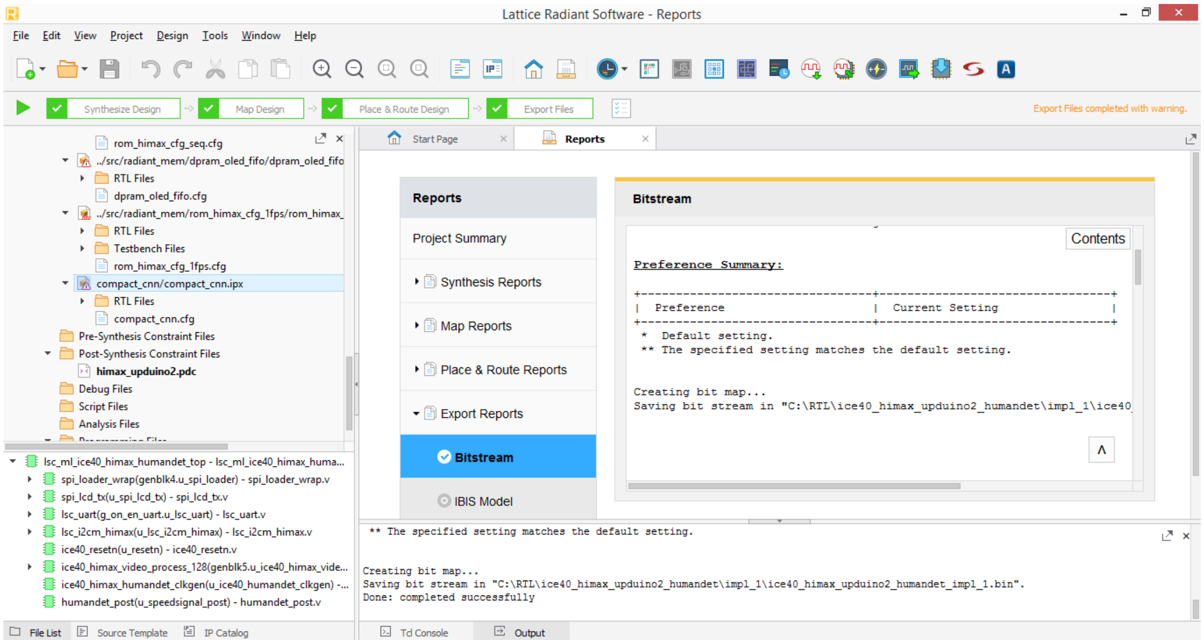

Figure 8.4. Radiant Software – Bitstream Generation Export Report

9. Running the iCE40 Human Presence Detection Demo

9.1. Functional Description

Figure 9.1 shows the diagram of the Human Presence demo. The camera captures the image data and sends it to the iCE40 UltraPlus device. iCE40 UltraPlus then uses the image data with the firmware file from the external SPI Flash to determine the outcomes.

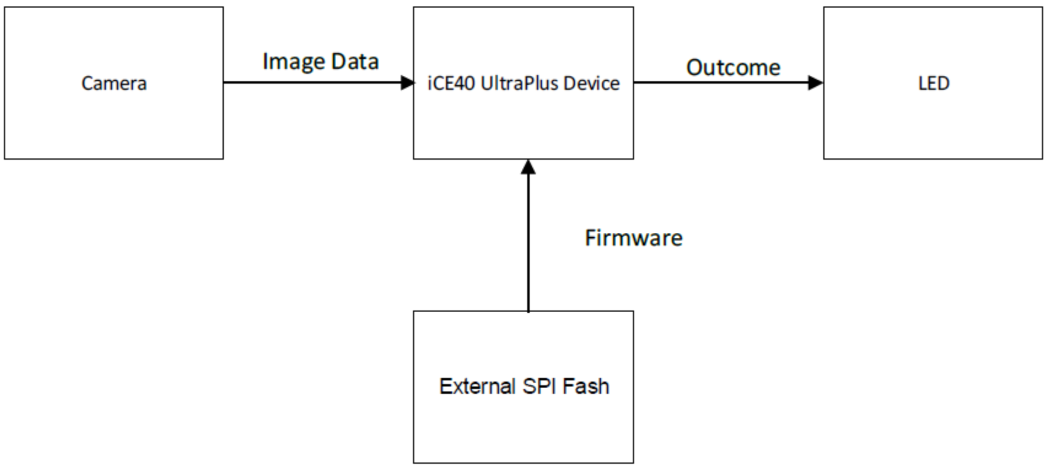

Figure 9.1. iCE40 Human Presence Demo Diagram

9.2. Programming Human Presence Detection Demo on iCE40 SPI Flash

This section provides the procedure for programming the SPI Flash on the HiMax HM01B0 UPduino Shield Board.

Two different files should be programmed into the SPI Flash. These files are programmed to the same SPI Flash, but at different addresses:

- Bitstream
- Firmware

To program the SPI Flash in Radiant Programmer:

1. Connect the HiMax HM01B0 UPduino Shield board to the PC using a micro USB cable. Note that the USB connector on board is delicate, so handle it with care.
2. Start Radiant Programmer.
3. In the Radiant Programmer **Getting Started** dialog box, select **Create a new blank project**.
4. Click **OK**.

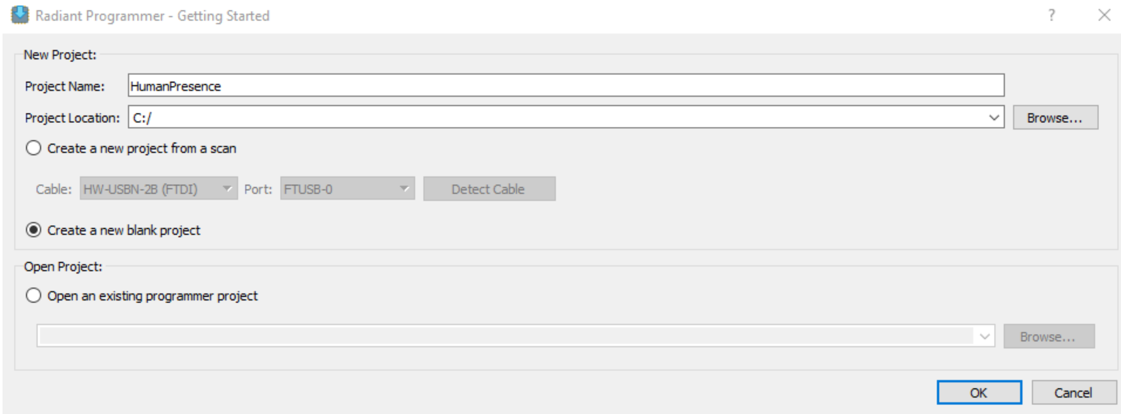

Figure 9.2. Radiant Programmer – Creating New Project

5. In the Radiant Programmer main interface, set **Device Family** to **iCE40 UltraPlus**.

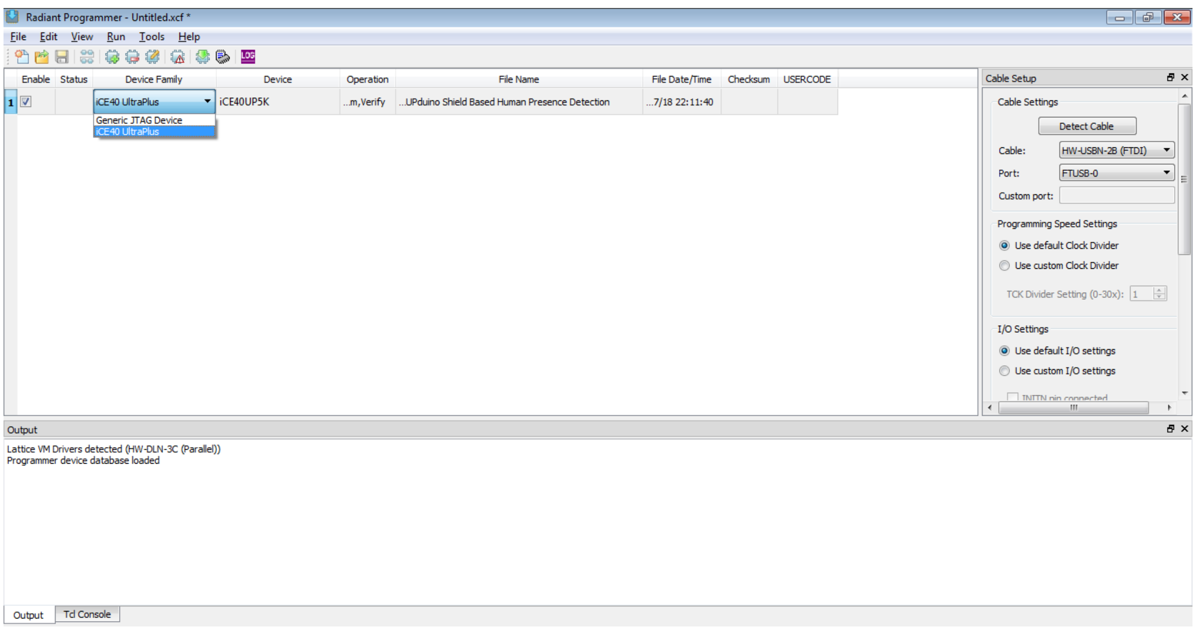

Figure 9.3. Radiant Programmer – iCE40 UltraPlus Device Family Selection

6. Set **Device** to **iCE40UP5K**.

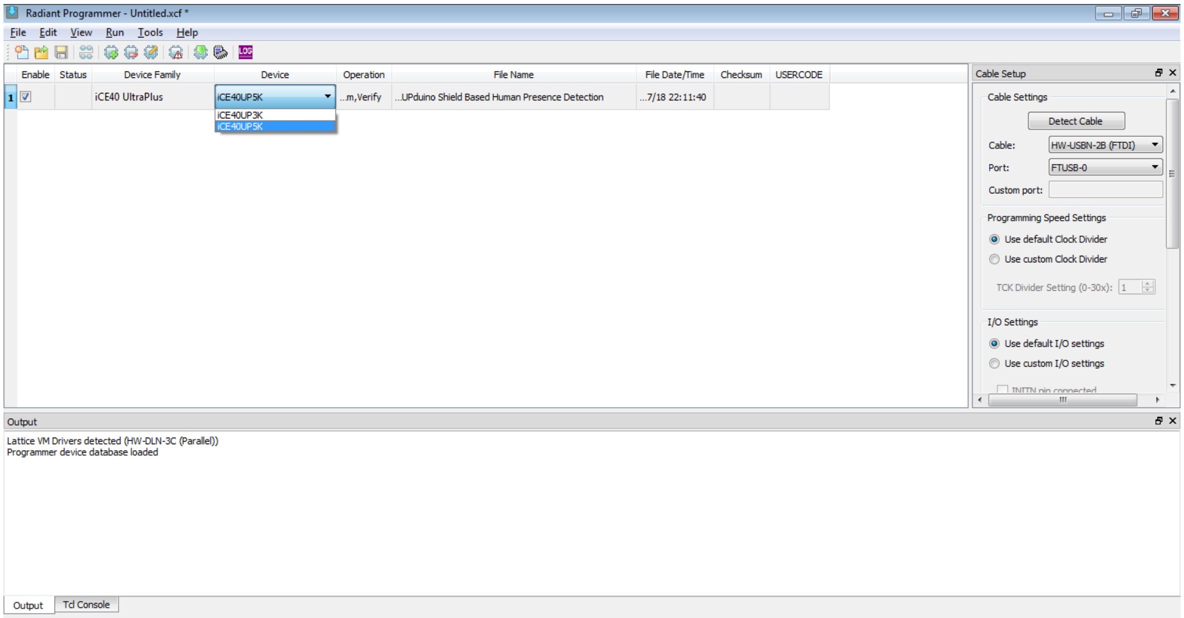

Figure 9.4. Radiant Programmer – iCE40 UltraPlus Device Selection

7. Select the iCE40 UltraPlus row and select **Edit > Device Properties**.
8. In the **Device Properties** dialog box, apply the settings below that are common to the two files to program.

   Under **Device Operation**, select the options below:
   - **Target Memory** – **External SPI Flash Memory**
   - **Port Interface** – **SPI**
   - **Access Mode** – **Direct Programming**
   - **Operation** – **Erase, Program, Verify**

   Under **SPI Flash Options**, select the options below:
   - **Family** – **SPI Serial Flash**
   - **Vendor** – **Winbond**
   - **Device** – **W25Q32**
   - **Package** – **8-pin SOIC**
9. To program the bitstream file, select the options below as shown in Figure 9.5.
   - Under **Programming Options**, select the human presence detection bitstream file in **Programming file**.
   - Click **Load from File** to update the **Data file size (Bytes)** value.
   - Ensure that the following addresses are correct:
     - **Start Address (Hex) – 0x00000000**
     - **End Address (Hex) – 0x00010000**

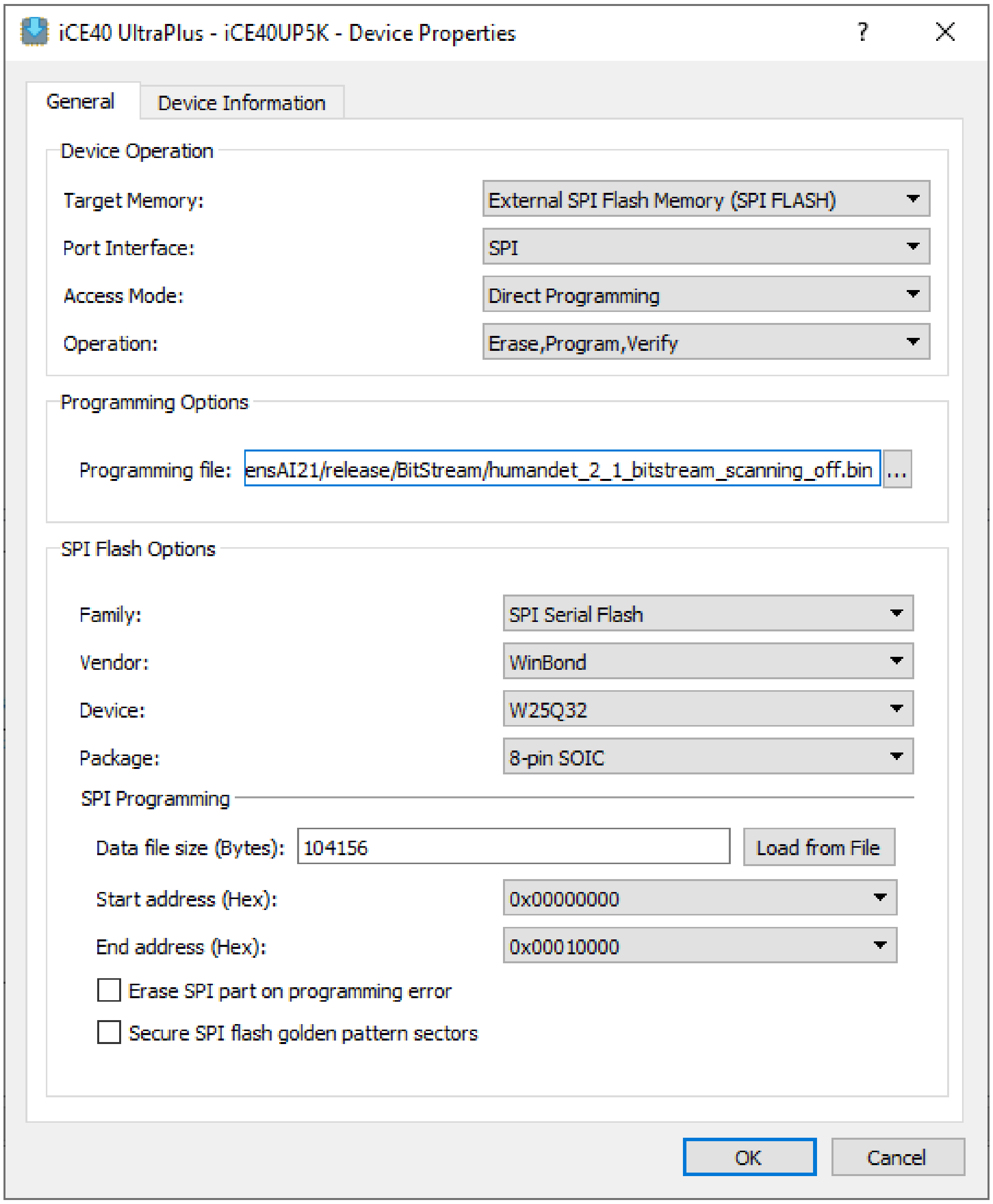

Figure 9.5. Radiant Programmer – Bitstream Flashing Settings

10. Click **OK**.
11. In the main interface, click **Program Device** to program the iCE40 human presence detection bitstream file.
12. To program the binary firmware file, select the options below as shown in Figure 9.6.
    - Under **Programming Options**, select the human presence detection firmware binary file in **Programming file**.
    - Click **Load from File**. Change **Data file size (Bytes)** value to **93140**.
    - Ensure that the following addresses are correct:
        - **Start Address (Hex) – 0x00020000**
        - **End Address (Hex) – 0x00030000**

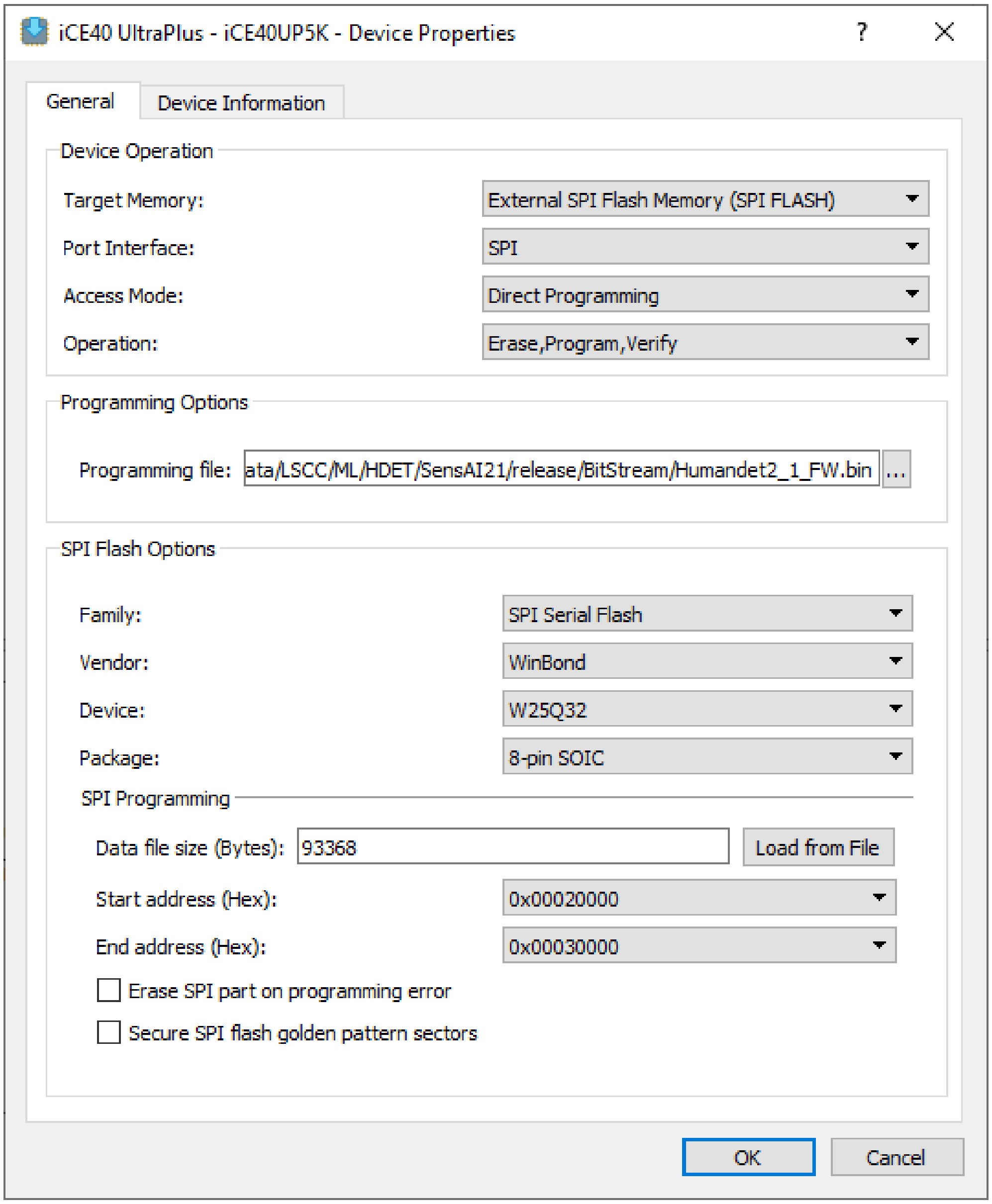

Figure 9.6. Radiant Programmer – Firmware Bin File Flashing Setting

13. Click **OK**.
14. In the main interface, click **Program Device** to program the binary file. After programming the files, perform a power cycle to start running the demo.

9.3. Running iCE40 Human Presence Detection Demo on Hardware

To run the demo and observe results on the board:

1. Power ON the HiMax HM01B0 UPduino Shield Board.

   Avoid any bright background.
2. Position a human in front of the camera. Based on MIRROR_MODE configuration led turns on. For MIRROR_MODE=0 led turns on based on MIRROR_MODE=0 (Zoning Enabled) section and for MIRROR_MODE=1 led turns on based on MIRROR_MODE=1 (Zoning Disabled) section.
3. An LED light turns on if a human is detected in its section. Note that Upper Left is the camera's Upper Left.

   Refer to Figure 9.7. for the location of camera and LED lights.

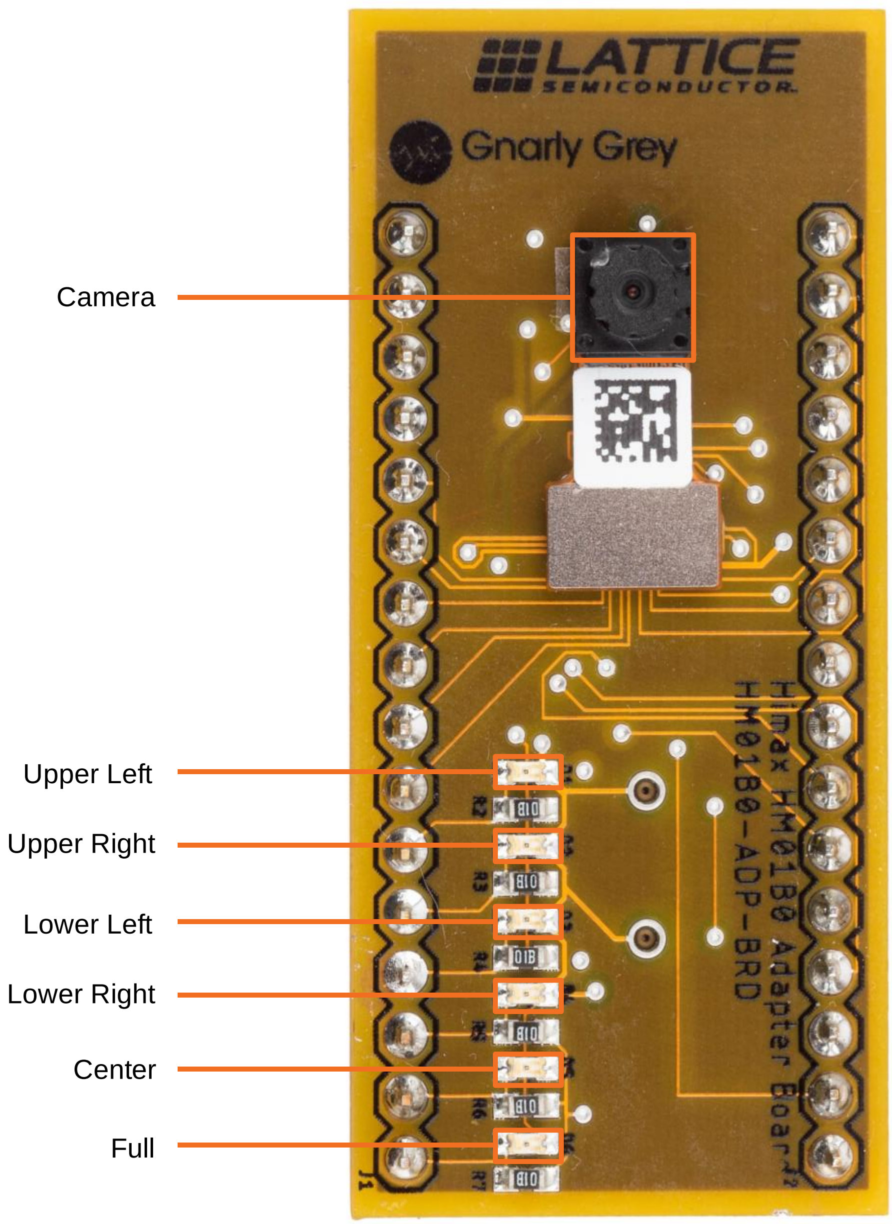

Figure 9.7. Camera and LED Location

Appendix A. Other Labelling Tools

Table A.1 provides information on other labelling tools.

Table A.1. Other Labelling Tools

Software	Platform	License	Reference	Converts To	Notes
annotate-to-KITTI	Ubuntu/Windows (Python based utility)	No License (Open source GitHub project)	https://github.com/SaiPrajwal95/annotate-to-KITTI	KITTI	Python based CLI utility. Just clone it and launch. Simple and Powerful.
LabelBox	JavaScript, HTML, CSS, Python	Cloud or On-premise, some interfaces are Apache-2.0	https://www.labelbox.com/	json, csv, coco, voc	Web application
LabelMe	Perl, JavaScript, HTML, CSS, On Web	MIT License	http://labelme.csail.mit.edu/Release3.0/	xml	Converts only jpeg images
Dataturks	On web	Apache License 2.0	https://dataturks.com/	json	Converts to json format but creates single json file for all annotated images
LabelImg	ubuntu	OSI Approved:: MIT License	https://mlnotesblog.wordpress.com/2017/12/16/how-to-install-labelimg-in-ubuntu-16-04/	xml	Need to install dependencies given in reference
Dataset_ annotator	Ubuntu	2018 George Mason University Permission is hereby granted, Free of charge	https://github.com/omenyayl/dataset-annotator	json	Need to install app_image and run it by changing permissions

References

- Google Tensorflow Object Detection Github
- Pretrained TensorFlow model for object detection
- Python sample code for custom object detection
- Train model using TensorFlow

Technical Support Assistance

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

Revision History

Revision 2.1, October 2019

Section	Change Summary
All	Changed document title from *Human Presence Detection Using Compact CNN* to *Human Presence Detection Using Compact CNN Accelerator IP*.
Preparing the Dataset	Added Data Augmentation section.
Training the Machine	Updated Neural Network Architecture and Training from Scratch and/or Transfer Learning section.
Creating Frozen File	Removed Generating pbtxt File section.
Creating Binary File with SensAI	Updated figures.
Hardware (RTL) Implementation	Newly added section.
Running the iCE40 Human Presence Detection on Demo	Updated figures in Programming Human Presence Detection Demo on iCE40 SPI Flash and Running iCE40 Human Presence Detection Demo on Hardware section.

Revision 1.0, May 2019

Section	Change Summary
All	Initial release.

www.latticesemi.com